UNIVERSITAT POLITÈCNICA DE VALÈNCIA

STUDY, ANALYSIS AND NEW SCHEDULING PROPOSALS IN PARTITIONED REAL-TIME SYSTEMS.

PhD Thesis by Ana Guasque Ortega

PhD Program in Automation, Robotics and Industrial Computer Science

Directed by Dr. Patricia Balbastre Betoret
and Dr. Alfons Crespo Lorente

December of 2019

Study, analysis and new scheduling proposals in partitioned real-time systems.

Author: Ana Guasque Ortega.

Directors: Dr. Patricia Balbastre Betoret and Dr. Alfons Crespo Lorente.

Tutor: Dr. Patricia Balbastre Betoret.

Text printed in Valencia

First edition, December 2019

Abstract

In our everyday lives, more and more computers are controlling our environment: mobile phones, industrial processes, driving assistance, etc. All these systems present strict requirements to ensure proper behaviour. In many of these systems, the time at which the action is delivered is as important as the logical result of the computation. About 40 years ago, real-time systems began to attract attention in computing field and nowadays are applied in wide ranging areas as industrial applications, aerospace, telecommunication applications, consumer electronics, etc.

Some real-time challenges that must be addressed are determinism and predictability of the temporal behaviour of the system. In this sense, to guarantee program execution and system response times are essential requirements that must be strictly met through appropriate task scheduling strategies.

Furthermore, multiprocessor architectures are becoming more popular due to the fact that processing capabilities and computational resources are increasing. A recent study estimates that there is an increasing tendency among multiprocessor architectures to combine different levels of criticality in the same system. In this sense, to provide isolation between applications is extremely required. Partitioned technology is able to deal with this purpose.

In addition, energy management is a relevant problem in real-time systems. Many real-time embedded systems, as wearable devices or mobile robots that require batteries, seek to find techniques that reduce the energy consumption and, as a consequence, increase the lifetime

of their batteries. Also clear operational, financial, monetary and environmental gains are reached when minimizing energy consumption.

Faced with all this, this work addresses the problem of schedulability and contributes to the study of new scheduling techniques in partitioned real-time systems. These techniques provide the minimum time to feasible schedule tasks sets. Moreover, allocation techniques for multicore systems whose main objective is to reduce the energy consumption of the overall system are also proposed.

Finally, some of the obtained results are discussed as conclusions and future works are introduced.

Resumen

En nuestra vida cotidiana, cada vez más ordenadores controlan nuestro entorno: teléfonos móviles, procesos industriales, asistencia a la conducción, etc. Todos estos sistemas presentan requisitos estrictos para garantizar un comportamiento adecuado. En muchos de estos sistemas, cumplir con las restricciones de tiempo es un factor tan importante como el resultado lógico de los cálculos. Desde hace aproximadamente 40 años, los sistemas en tiempo real son muy atractivos en el campo de la computación y hoy en día se aplican en áreas de gran alcance como aplicaciones industriales, aplicaciones aeroespaciales, telecomunicaciones, electrónica de consumo, etc.

Algunos retos a abordar en el campo del tiempo real son el determinismo y la predecibilidad del comportamiento temporal del sistema. En este sentido, garantizar la ejecución del programa y los tiempos de respuesta del sistema son requisitos esenciales que deben cumplirse estrictamente a través de estrategias apropiadas de planificación de tareas.

Además, las arquitecturas multiprocesador se están volviendo más populares debido al hecho de que las capacidades de procesamiento y los recursos computacionales de los sistemas están aumentando. Un estudio reciente estima que existe una tendencia creciente entre las arquitecturas multiprocesador a combinar diferentes niveles de criticidad en el mismo sistema. En este sentido, proporcionar aislamiento entre las aplicaciones es extremadamente necesario. La tecnología particionada es capaz de lidiar con este propósito.

Además, la gestión de la energía es un problema relevante en los sistemas en tiempo real. Muchos sistemas empotrados de tiempo real, como

dispositivos portátiles o robots móviles que requieren baterías, buscan encontrar técnicas que reduzcan el consumo de energía y, como consecuencia, aumenten la vida útil de sus baterías. También se obtienen claros beneficios operativos, financieros, monetarios y ambientales al minimizar el consumo de energía.

Con todo ello, este trabajo aborda el problema de planificabilidad y contribuye al estudio de las nuevas técnicas de planificación en sistemas particionados de tiempo real. Estas técnicas proporcionan el tiempo mínimo para planificar de manera factible conjuntos de tareas. Además, se proponen técnicas de asignación para sistemas multiprocesador cuyo objetivo principal es reducir el consumo de energía del sistema global.

Finalmente, se presentan los resultados obtenidos así como los trabajos futuros relacionados con este trabajo.

Resum

En la nostra vida quotidiana, cada vegada més ordenadors controlen el nostre entorn: telèfons mòbils, processos industrials, assistència a la conducció, etc. Tots aquests sistemes presenten requisits estrictes per a garantir un comportament adequat. En molts d'aquests sistemes, complir amb les restriccions de temps és un factor tan important com el resultat lògic dels càlculs. Des de fa aproximadament 40 anys, els sistemes en temps real són molt atractius en el camp de la computació i hui dia s'apliquen en àrees de gran abast com a aplicacions industrials, aplicacions aeroespacials, telecomunicacions, electrònica de consum, etc.

Alguns reptes a abordar en el camp del temps real són el determinisme i la predictibilitat del comportament temporal del sistema. En aquest sentit, garantir l'execució del programa i els temps de resposta del sistema són requisits essencials que han de complir-se estrictament a través d'estratègies apropiades de planificació de tasques.

A més, les arquitectures multiprocessador s'estan tornant més populars a causa del fet que les capacitats de processament i els recursos computacionals dels sistemes estan augmentant. Un estudi recent estima que existeix una tendència creixent entre les arquitectures multiprocessador a combinar diferents nivells de criticitat en el mateix sistema. En aquest sentit, proporcionar aïllament entre les aplicacions és extremadament necessari. La tecnologia particionada és capaç de bregar amb aquest propòsit.

A més, la gestió de l'energia és un problema rellevant en els sistemes en temps real. Molts sistemes embebits de temps real, com a dispositius portàtils o robots mòbils que requereixen bateries, busquen trobar

tècniques que reduïsquen el consum d'energia i, com a conseqüència, augmenten la vida útil de les seues bateries. També s'obtenen clars beneficis operatius, financers, monetaris i ambientals en minimitzar el consum d'energia.

Amb tot això, aquest treball aborda el problema de planificabilitat i contribueix a l'estudi de les noves tècniques de planificació en sistemes particionats de temps real. Aquestes tècniques proporcionen el temps mínim per a planificar de manera factible conjunts de tasques. A més, es proposen tècniques d'assignació per a sistemes multiprocessador l'objectiu principal del qual és reduir el consum d'energia del sistema global.

Finalment, es presenten els resultats obtinguts així com els treballs futurs relacionats amb aquest treball.

Acknowledgements

When I look back and remember how this road began, I find myself suffering in the last year of my degree. There I studied the first subject on "real time", which at that time I was still not sure about what that was. There I met Patricia and Alfons, my professors of that subject and now directors, and I can say that from then on, everything changed. They gave me the opportunity to do the final project under their supervision and that was the beginning of the cycle that closes today with this doctoral thesis. So my first thanks are to them, that with their patience, help and trust placed in me, they have made me grow as a person and as a researcher. I am immensely grateful for all that I have achieved thanks to them.

I would also like to thank all the teachers that I have been able to count on throughout this time, for solving academic questions or for a pleasant chat time. Thanks, especially, to Pepe and Gabriela. Of course, thanks to the people who work in the institute, which manage all the paperwork and make all administrative formalities easier. Thanks, Mercedes. Thanks also to Carlos Torras for all his technical help in the department.

I would not want to forget the people I met during my stay in Kaiserslautern: Gerhard, thanks to whom I learned that a long meeting became more bearable with a beer in the kitchen; Markus, for his technical help and his passion for Andalusia and Cruzcampo beer; to Kristin, my only girl colleague and thanks to whom I learned so much about German culture; Gautam, for making me feel integrated into the group since I arrived; Florian, for being the best office partner I could have; Rodrigo, for his charisma and understanding; and to Ali and Ankit for

their talks and advices. Thanks to Stephanie for her management and for making my stay easier all the time. Of course, thanks to all the friends I made there, especially Astrid and Ana.

During these years, many colleagues have passed through this laboratory from which I write these words today. I can say that a large part of them stopped being colleagues to become friends and I know they will continue to be in spite of distance and time. Experience has taught me that, if I do not name them in the acknowledgments section, they will remind it until my last days so thank you very much for your time of meals, coffees, talks, beers, farewells and even weddings: from Manu (You're already a dad!), Tomás *man*, Edu *chino*, Jose Luis *chanchullos*, Albert *el escritor*, Lorena *pelazo*, Toni *Phi Phi Islands*, Nico *nuevegag*, Iván *el trolaso*, Andrés *paquetitos*, Jose Manuel *el murciano*, Savi and her contagious joy and enthusiasm to Andreu and our conversations about life. From all of them I have great memories. Thank you for being with me in the good times and, above all, in the bad times.

Likewise, I want to express my gratitude to all my friends, whom I cannot quote one by one because I would leave some. Thank you for all these years of support and friendship.

Finally, I want to remember my greatest support: my family. Thank you for all your help and energy, from the first day. To my parents, for their dedication and encouragement in all the moments we have been through; to my sister, that although we are like night and day, we will always be there for an advice and will always have a shoulder to cry on; and to Paco, for his patience and support during all these years.

Thank you all.

Thanks,

Ana Guasque

December 2019

Agradecimientos

Cuando echo la vista atrás y recuerdo como empezó este camino, me encuentro a mí misma sufriendo en el último año de carrera. Allí cursé la primera asignatura sobre "tiempo real", que por aquel entonces todavía no tenía muy claro qué era aquello. Ahí conocí a Patricia y Alfons, mis profesores de la asignatura y ahora directores, y puedo decir que a partir de entonces, todo cambió. Ellos me dieron la oportunidad de hacer el proyecto final de carrera bajo su supervisión y ese fue el inicio del ciclo que hoy se cierra con esta tesis doctoral. Así que mi primer agradecimiento es para ellos, que con su paciencia, ayuda y confianza depositada en mí, me han hecho crecer como persona y como investigadora. Les estoy inmensamente agradecida por todo lo que he conseguido hasta ahora gracias a ellos.

También quisiera dar las gracias a todos los profesores con los que he podido contar a lo largo de este tiempo, ya sea para una duda a nivel académico o para un rato de charla agradable. Gracias, en especial, a Pepe y Gabriela. Por supuesto, gracias a la gente que trabaja en el instituto, que nos gestiona todos los "papeleos" y nos hace más fáciles todos los trámites administrativos. Gracias, Mercedes. Gracias también a Carlos Torras por toda su ayuda técnica en el departamento.

No quisiera olvidarme de la gente que conocí durante mi estancia en Kaiserslautern: a Gerhard, gracias al que aprendí que una reunión larga se hacía más llevadera con una cerveza en la cocina; a Markus, por su ayuda técnica y su pasión por Andalucía y la cerveza Cruzcampo; a Kristin, mi única compañera chica y gracias a la que aprendí tanto sobre la cultura alemana; a Gautam, por hacer que me sintiera integrada en el grupo desde que llegué; a Florian, por ser el mejor compañero de

despacho que pude tener; a Rodrigo, por su carisma y comprensión; y a Ali y Ankit por sus charlas y consejos. Gracias a Stephanie por su gestión y por facilitarme la estancia durante todo el tiempo. Por supuesto, gracias a todos los amigos que hice allí, en especial, a Astrid y Ana.

Durante estos años, han pasado muchos compañeros por este laboratorio desde el que hoy escribo estas palabras. Puedo decir que gran parte de ellos dejaron de ser compañeros para convertirse en amigos y sé que lo seguirán siendo a pesar de la distancia y del tiempo. La experiencia me ha enseñado que, si no los nombro en los agradecimientos, me lo van a recordar hasta mis últimos días así que muchísimas gracias por sus ratos de almuerzos, comidas, cafés, charlas, cervezas, despedidas e incluso bodas: desde Manu (¡ya eres papá!), Tomás *man*, Edu *chino*, Jose Luis *chanchullos*, Albert *el escritor*, Lorena *pelazo*, Toni *Phi Phi Islands*, Nico *nuevegag*, Iván *el trolaso*, Andrés *paquetitos*, Jose Manuel *el murciano*, Savi y su alegría y entusiasmo contagiosos hasta Andreu y nuestras conversaciones sobre la vida. De todos ellos me llevo buenísimos recuerdos. Gracias por estar conmigo en los buenos y, sobre todo, en los malos momentos.

Así mismo, quiero expresar mi agradecimiento a todos mis amigos, a quienes no puedo citar de uno en uno porque me dejaría a alguno. Gracias por todos estos años de apoyo y amistad.

Para acabar, quiero acordarme de mi mayor apoyo: mi familia. Gracias por toda vuestra ayuda y energía, desde el primer día. A mis padres, por su dedicación y ánimo en todos los momentos que hemos pasado; a mi hermana, que aunque seamos como la noche y el día, siempre estaremos para un consejo y para ofrecernos un hombro donde apoyarnos; y a Paco, por su paciencia y apoyo durante todos estos años.

Gracias a todos.

Muchas gracias,

Ana Guasque

December 2019

Contents

CHAPTER

1

Introduction

Lifestyle changes day by day and, consequently, new technological requirements emerge to adapt to human exigencies. In recent years, the importance of real-time embedded systems is increasing sharply in robotic industry, air traffic control systems, process control systems, avionics, etc.

Main characteristic that distinguish real-time systems from others is time constraint. In these systems, not only the system response is important but also meeting deadlines. Due to the importance of temporal requirements, this work aims to provide an study and analysis of scheduling policies in this kind of systems.

1.1 Motivation

Embedded systems appear in our everyday life: personal home affairs, process automatization in industries, automotive sector, entertainment, avionics, etc. In the major part of these systems, real-time features are indispensable. The first real-time operative system appeared more than 30 years ago and, since then, they have become essential in human affairs.

Real-time systems require a valid response within finit temporal constraints. In case of these requirements are not fullfilled, undesired consequences will be caused in the system. Attending to the level of temporal exigency, real-time systems are divided into hard real-time systems or soft real-time systems. On the one hand, hard real-time systems are those in which missing any temporal constraint may supposse catastrophic results. On the other hand, missing deadlines in a soft real-time system may produce damage or poor global operation but no grave damages. As systems have always had soft and hard components, achieving isolation between them is a constant over time. Thanks to the isolation between components, the systems reduce their complexity, avoid error propagations and reduce development and certification costs. Distributed systems were, and continue to be, an architecture that allow these requirements to be reached. Since 2007 to the present, an increasingly trend in the design of real-time systems is to integrate components with different levels of criticality (or relevance) in a hardware platform. They are known as mixed criticality systems. Requirements as cost, space, power consumption, etc. make most of the complex embedded systems are envolved into mixed criticality systems. The main fields in which mixed criticality systems are involved are avionics, space and automotive. For example, in avionics, to ensure the aircraft's control, flight software control must execute accurately with strong deadlines.

At the same time, as modern systems have increased their computation capacity demands, hardware industry is migrating from single cores to multicore architectures. Multi-core processors emerged from the necessity of enhancing the computation capacity in an increasingly compact hardware. Aditionally, they cope with the necessity of reducing power consumption and heat dissipation without decreasing the system operating frequency and therefore jeopardizing performance.

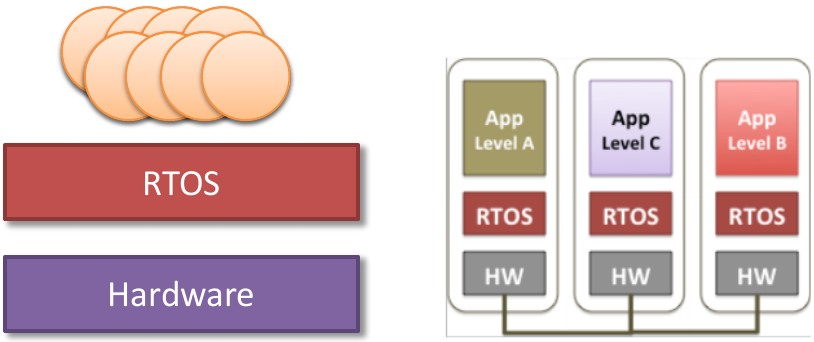
However, these architectures have introduced many challenges in maximizing application performance and best using the available processing capacity.

New processors not only improve functionalities and enhance possibilities for embedded applications, but increase their complexity. Sharing resources between applications may result in difficulties and interferences that might hinder the logical design and the performance of the system execution. From this point, the protection between applications is needed. This gave rise to partitioned systems, developed to address security and safety problems. Spatial and temporal isolation properties of the partitioned architectures are very relevant aspects in partitioned systems. Spacial isolation implies that each application can only access to its independent memory addresses. In monocore, temporal isolation means only one application at a point of time has access to the system resources, whereas is not possible to an application run when another application is running. It requires the use of efficient schedulability techniques.

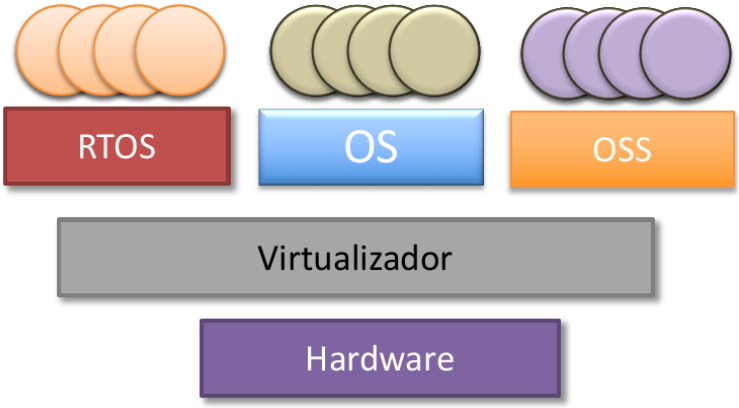
Figure 1.1 shows the evolution from embedded systems (a), with lots of tasks and functionalities, which are hard to debug and maintain, to distributed systems (b), which reduce the complexity and provide isolation between applications. With the emergence of mixed criticality systems, partitioned architectures appeared (c), which also provide isolation between applications in a virtualized hardware.

Schedulability analysis in real-time systems is a extended research area, specially in single core systems. However, increasing the number of cores per system also increases the complexity of the system in terms of schedulability. Since the first commercial dual core processor chip, the POWER4, developed by IBM in 2001, the interest in real-time multiprocessor scheduling has increased significantly. However, there are many open issues and topics that have to be updated: scheduling problem, core allocation, energy savings, etc.

These topics have been present in the lines of research in the group of industrial computer science and real-time systems in Universitat Politècnica de València, in which I take an active part. A closer look into the group trajectory through international, national and autonomous projects, from 2002 to the present day, shows that the first European project with more relevance was OCERA [**?**], in which this group worked as a coordinator.



(a) Embedded system (b) Distributed system



(c) Partitioned system

Figure 1.1: From embedded to partitioned systems

The main objective of project OCERA, Open Components for Embedded Real-time Applications was the design and implementation of a library of free software components for the design of embedded real-time systems.The components were designed to cover the widest application range including fully critical systems and systems with different critically degrees.

Next project emerged in 2006 and was called FRESCOR [**?**], Framework for Real-time Embedded Systems based on COntRacts. The main objective of this project was to develop the technology and infrastructure required to effectively use the most advanced techniques in real-time applications with flexible scheduling requirements. This methodology was well suited to address very dynamic systems, such as those based on reconfigurable architectures.

The virtualization technology around the hypervisor was initially developed in OCERA and improved in FRESCOR. In these projects, x86 architecture was the used platform. In Multi-cores Partitioning for Trusted Embedded Systems (MultiPARTES) [**?**] the virtualization techniques were extended to multicore systems for x86 and sparcV8 architectures, composing a heterogeneous system. The main challenge of this project (2011-2014) was supporting mixed criticality embedded systems on multicore open source virtualized platforms in such a way that the development, validation and certification efforts can be lower than the corresponding effort required on independent hardware platforms when using an appropriate methodology. The starting point for the virtualization support is XtratuM, a cost-effective open source hypervisor developed specifically for critical embedded systems by our group.

At the same time, in 2012, the project High-Integrity Partitioned Embedded Systems (Hi-Partes) [**?**] emerged. Its objective was to improve the methods and technology of high-integrity embedded systems based on virtualization.

In the project Distributed REal-Time Architecture for Mixed Criticality Systems (DREAMS) [**?**], 2013-2017, the goal was to develop a cross-domain architecture and design tools for networked complex systems, where application subsystems of different criticality are supported. DREAMS will deliver architectural concepts, meta-models, virtualization technologies, model-driven development methods, tools, adaptation strategies and validation, verification and certification methods for the seamless integration of mixed-criticality.

In 2016, a project that took part in H2020 Programme was SAFEPOWER [**?**]. Its main objective was to enable the development of mixed-criticality systems with low power, energy and temperature in combination with safety, real-time and security support by a reference architecture orchestrating different local power-management techniques. SAFEPOWER project is producing a reference architecture and implementing the platforms, complemented by analysis, simulation and verification tools, to deliver power savings of up to 50% on the computing systems embedded in such safety-critical systems.

The last project related to this field was Sistemas Ciber-fisicos de Criticidad Mixta sobre Plataformas Multinucleo, M2C2 [**?**], 2015-2017, whose objective was to contribute through methods and techniques to the development of cyber-physical systems, applied to multicore platforms, allowing mixed-criticality applications. This approach is also based in virtualization, in order to achieve temporal and spatial isolation and guarantee fault isolation.

This thesis is framed within the last two projects, in the field related to partitioned multicore real-time systems, specially in those working packages related to scheduling techniques and methods.

In addition to schedulability, energy management is a problem to be addressed in many real-time embedded systems as wearable devices, mobile robots, etc. Reducing the energy consumption involves increasing lifetime's batteries and this has a significant positive impact in the comfort and quality of human life.

To deal with this problem, two wide techniques are used: *Dynamic Voltage and Frequency Scaling* (DVFS) and *Dynamic Power Management* (DPM). DVFS techniques consists of decreasing the frequency and the voltage of the processor in order to reduce the overall energy consumption, without compromising deadlines. DPM approaches play on changing between active and inactive processor states, also guaranteeing deadlines of real-time tasks.

Despite there is a large number or proposed approaches regarding to energy management in real-time systems, most of them are focused on previous techniques or disregard real-time constraints.

Both schedulability and energy management in real-time systems have still many open questions and research challenges. This work presents a contribution in this field by exploring new approaches for partitioned real-time systems.

Once the context of this work has been defined, Figure 1.2 summarizes the areas related to this field that are subject to our research. Moreover, the chapters which concretely contribute to each of these areas are indicated.

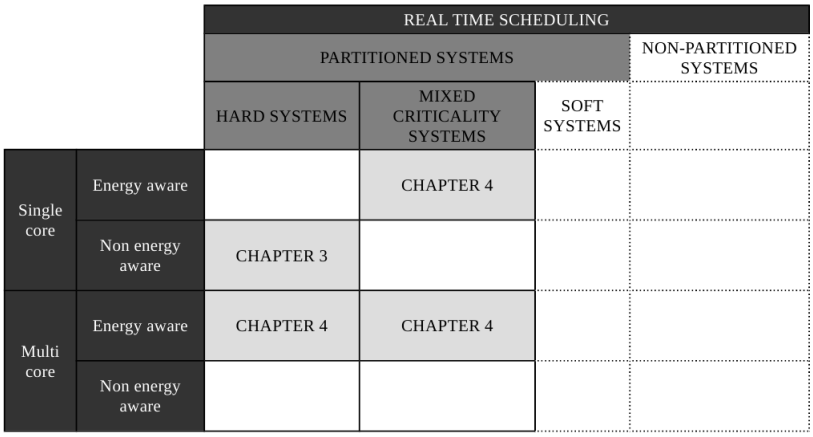


Figure 1.2: Taxonomy of real-time scheduling.

Real time scheduling can be roughly distinguished between partitioned systems and non-partitioned systems. Generally speaking, a partitioned system is one in which there is no migration of tasks between cores, in contrast to global systems in which task jobs can migrate between processors. However, in our case, the term "partitioned" refers to partitioned software architectures used within Integrated Modular Avionics, for example, in space applications, that are based on the ARINC-653 standard. Thanks to partitioned architectures, a strong isolation between processes is achieved.

Following the interests of the group of industrial computer science and real-time systems, partitioned systems and, specially, those with hard and mixed criticality real-time requirements are considered. As aerospace is one of our main research fields, our research deals with hard or mixed criticality real-time problems. Systems in which all tasks are soft real-time tasks as, for example, multimedia applications, are not considered in this thesis due to the fact that the problems we study present more stringent and complex requirements.

Chapter 3 contains our contributions about scheduling algorithms in single core real-time hard systems. Chapter 4 includes energy-aware considerations for both, hard and mixed criticality systems, and for single or multicore platforms.

1.2 Objectives

The main objective of the present work is to contribute to the development of scheduling techniques in partitioned and hierarchical real-time systems. After a comprehensive study of literature regarding to allocation and scheduling techniques in real-time systems, a number of issues remains to be tackled in partitioned architectures.

According to this, this work follows the next line of research, starting with fundamental scheduling concepts in real-time systems and ending with new optimal and feasible proposals. For this purpose, the following objectives are proposed:

- Review and analysis of the state of the art and key concepts for multi-core scheduling and allocation policies in real-time systems, specially in partitioned and hierarchical architectures. The extension of the study to energy-aware scheduling algorithms in these systems is also contemplated.
- Comprehensive analysis of the previous work of this contribution by presenting a shedulability analysis in the case of two-level hierarchical systems.
- To achieve the remaining challenge for hierarchical systems with unknown scheduling policies in the global level.
- Determination of schedulable functions that provide the minimum required time in a partitioned system in order to ensure the feasibility of the overall system.
- Development of allocation techniques on the basis of the results obtained previously in partitioned multi-core systems.
- To approach energy savings techniques in multi-core partitioned systems. Calculation of the optimum energy consumption through power, time-frequency and energy-time-frequency mathematical models.
- Evaluation of previous approaches by developing different simulators: heuristic algorithm and constraint programming methodologies. Comparison of the results.

- Validation of the approaches by applying theses algorithms in an experimental scenario.

1.3 Organization of this thesis

Proposed developments for achieving these objectives are addressed along this work. First, in Chapter 2 a detailed review of the main concepts and works related with scheduling policies in partitioned real-time systems are introduced. It includes those related to mixed criticality systems, offline techniques, allocation and energy management. Next, in Chapter 3, new scheduling techniques are proposed. This chapter also includes the evaluation of these algorithms and their validation. Then, in Chapter 4 energy savings techniques in multicore partitioned systems are introduced. Furthermore, an evaluation by means of the implementation of two simulators is presented: one based on heuristic algorithms and other based on a constraint programming framework. Finally, in Chapter 5, the obtained results are discussed as conclusions leading to the establishment of the future work is introduced.

CHAPTER

2

State of the Art

Nowadays, a great number of researches that aims to contribute to the development of scheduling techniques in real-time systems are found. This chapter reviews the key results in this field from its origins in the fifties to the present day.

2.1 Introduction

2.1.1 Real-time systems

A real-time system is "any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world and the output has to relate to that some movement. The lag from input time to output time must be sufficiently small for acceptable timeliness." [**?**] This is the definition provided by Oxford dictionary of computing, reviewed by a team of computer specialists. There are many other definitions of real-time systems ([**?**],[**?**] [**?**],[**?**]) but they all agree in the importance of temporal fulfilment in order to avoid an undesirable state of the system.

Predictability is an essential characteristic of a real-time system. Velocity and efficiency are not sufficient conditions for a system to be considered as a real-time system. An analysis of the response time of the system is required.

Whereas in real life events happen at the same time, computers work sequentially. Some of these events are unpredictable so the conflicts with sequential execution on the controller appear. Also in these situations the system has to produce the results logically and in the correct times.

Many real-time systems are control systems: avionics, process control, robotics, monitoring services, etc.

Real-time systems are classified into two distinct categories, depending on the consequences of non-compliance of temporal requirements: hard real-time systems and soft real-time systems[**?**].

- Hard or critical real-time systems are those in which temporal constraints must be met. Otherwise, their consequences could have a dramatic impact on human life, on the environment, on the system, etc. These systems have, at least, one hard deadline. Some examples of these kind of systems are aeronautics, satellites control, critical operations in robots, autopilot systems, etc.

- Soft real-time systems are those in which occasional deadline misses produce a degradation of their quality of service, but do not failures. Examples

of soft real-time systems: audio and video delivery software for entertainment, where failures are undesirable but not catastrophic. Data acquisition, production processes, etc. are other example of soft real-time systems.

The main differences between these systems are stated in Table 2.1.

Characteristic	Hard real-time	Soft real-time
Response Time	Hard-required	Soft-required
Peak-load performance	Predictable	Degraded
Control of pace	Environment	Computer
Safety	Often critical	Non-critical
Size of data files	Often critical	Non-critical
Redundancy type	Active	Checkpoint-recovery
Data integrity	Short-term	Long-term
Error detection	Autonomous	User assisted

Table 2.1: Major differences between hard and soft real-time systems

Throughout this chapter, different considerations about real-time systems are being explained. First, general scheduling techniques in real-time systems are considered. Then, some particular new trends as mixed criticality real-time systems, hierarchical systems or partitioned systems are introduced.

2.2 Scheduling policies in real-time systems

As stated before, real-time systems are characterised by the importance of meeting temporal requirements of tasks. Many papers and surveys have been published about scheduling in real-time systems [**?**], [**?**], [**?**].

A real-time process or application can be divided in a set of tasks. Each task can be executed on a regular basis (i.e. periodic task) or randomly (i.e. non-periodic task). Periodic tasks are composed by activations A_i every specified time, known as period (T_i). Non-periodic tasks are released as a result of an external event. If there is a minimum inter-arrival time between activations of a non-periodic task, it is said to be sporadic task. Otherwise, it is considered aperiodic.

Each activation is defined by its temporal requirements. Usual parameters of an activation *i* are:

- Release time (r_i) is the earliest time when an activation is ready for start the execution.

- Computation time (C_i) is the time spent executing its actions. Usually, in hard or soft real-time systems, worst case execution time (WCET) is considered. This term is necessary to perform the schedulability analysis, to ensure that deadlines are met and to evaluate the resource needs under all conditions in real-time systems. The WCET accuracy is a safety-critical problem and its analysis has been studied in lots of works [**?**], [**?**] and international workshops[1].

- Deadline (D_i) is the time by which the execution of the activation is required to complete. If the activation ends before completing the deadline, the activation will be schedulable. Therefore, the issue of meeting deadlines is one of the most important aspects in the real-time system scheduling. From now on, D_i is the deadline or relative deadline, in contrast with absolute deadline, which is $k \cdot T_i + D_i$ for the activation A_k in a periodic task *i*.

[1] International workshop in Worst-Case Execution Time analysis, last edition: https://wcet2018.wp.imt.fr/

- Response time (rt_i) is the time between an activation is released and the end of its execution.
- Offset (ϕ_i) is time between the start time and the time when first activation starts.

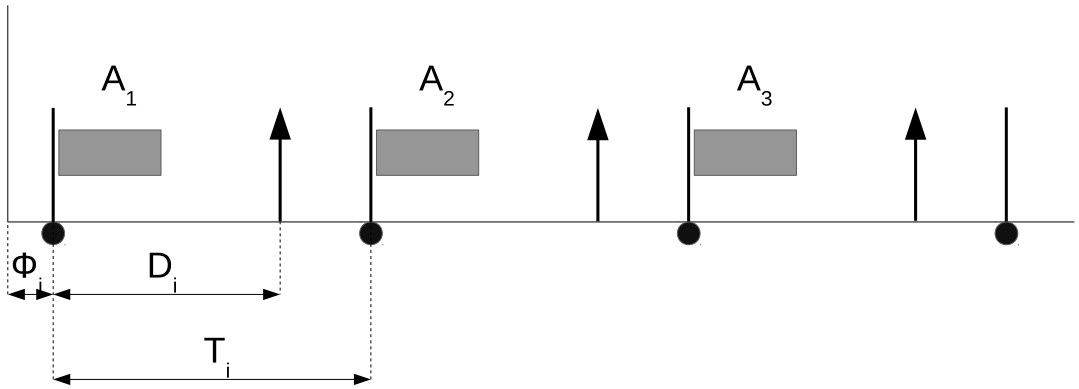


Figure 2.1: Temporal parameters of a periodic task.

Figure 2.1 shows temporal parameters of a task.

The use of effective and accurate schedulability analysis techniques is required in order to accomplish temporal requirements in a real-time system, while guaranteeing system performance. In these systems, the order of execution of tasks is relevant and the statement of an order that ensure the feasibility of the task set is called schedulability. A schedulability process takes place in two phases:

- Selection of the scheduling algorithm and analysis of the schedulability of the task set with this algorithm.
- Generation of the scheduling to ensure that temporal requirements are satisfied.

Terms like “feasibility" and “schedulability" are now defined in order to clarify them [**?**]: A task set (or application) is said to be *feasible* with respect to a given system if there exists some scheduling algorithm that can schedule all possible sequences of jobs that may be generated by the task set on that system without missing any deadlines. A scheduling algorithm is said to be *optimal* with respect

to a system and a task model if it can schedule all of the task sets that comply with the task model and are feasible on the system.

A task is *schedulable* according to a given scheduling algorithm if its worst-case response time under that scheduling algorithm is less than or equal to its deadline. Similarly, a task set is referred to as schedulable according to a given scheduling algorithm if all of its tasks are schedulable. A scheduling algorithm is referred to as *optimal* if it can schedule all of the task sets that can be scheduled by any other algorithm, that is, all of the feasible task sets.

Selecting the scheduling policy for a real-time system depends on factors as number of processors, precedence between tasks, etc. A classification of schedulers is depicted in Figure 2.2.

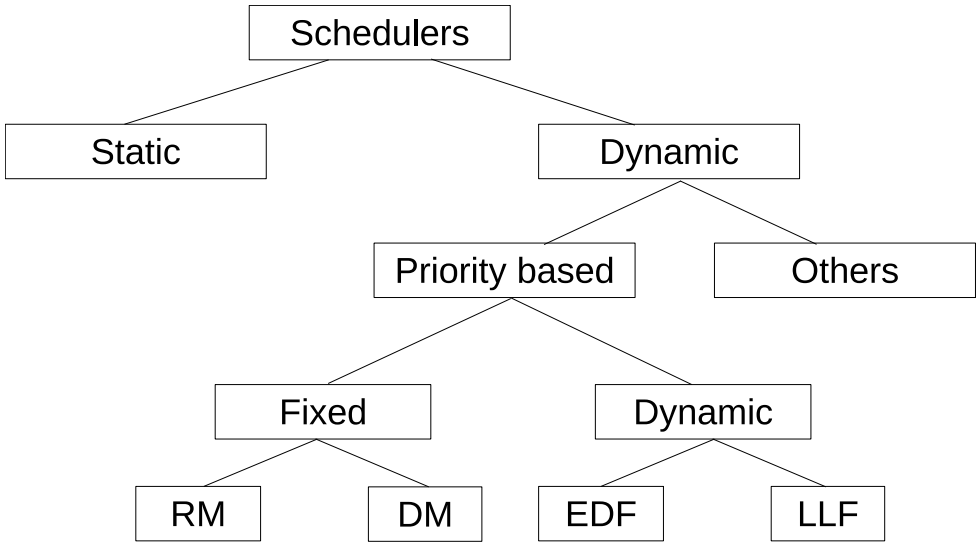


Figure 2.2: Real-time schedulers classification.

Roughly, scheduling real-time algorithms are classified in static or dynamic.

2.2.1 Static schedulers

Static scheduling, on one hand, requires *a priori* knowledge of the characteristics of the tasks. The static scheduler generates offline a sequence of tasks executions, called plan, and it is cyclically repeated. This scheme is known as cyclic executive [**?**], [**?**]. This static plan is saved in a table and indicates the moment of time

in which each task must be executed, in such a way that the scheduler should only follow the indications of the table. The verification of the schedulability using this strategy must be carried out during the construction of the plan. Static scheduling presents advantages as low cost in run time but also disadvantages as lack of flexibility and good knowledge of the task set.

2.2.2 Dynamic schedulers

Dynamic scheduling, on the other hand, determines the tasks execution order at runtime, considering the characteristics of the tasks and the state of the system. Dynamic schedulers are more flexible and achieve higher processor utilization values than static schedulers. However, the overhead attributable to dynamic scheduling is greater. Among dynamic schedulers, priority based schedulers exist. They ensure proper timing behaviour and predictability of the system. Priority is the criterion to select a task between those ready to be executed. Then, CPU is in charge to execute the runnable task with higher priority in the system.

An important issue in dynamic scheduling is the decision taken by the scheduler regarding to the state and priority of the tasks. A task will be denoted as preemptive if it can be interrupted by other tasks and restarted later. A non-preemptive task is executed until is completed, without interruption. If any of the tasks of the system is preemptive, the scheduler will also be preemptive.

Priority based schedulers are subdivided into fixed and dynamic priority schedulers.

2.2.2.1 Fixed priority based schedulers

In this scheme, the scheduler assigns an initial priority to the tasks and it remains constant during all the execution. In this scheme, changes on the task set are immediately taken into account by the scheduler. Other advantages are that sporadic tasks are easily accommodated and the behaviour is deterministic on overloads, being the lower priority tasks the most affected. On the contrary, a kernel that supports fixed priorities is required.

Among fixed priority scheduling (FPS), Rate Monotonic and Deadline Monotonic are the most well-known approaches. Rate Monotonic Scheduling (RMS)

assigns highest priority to the task with shortest period. This seminal algorithm, developed by Liu and Layland [**?**], assumes preemptive independent tasks with deadlines equal to periods. In some cases, tasks may have large periods but require a short response time. In these situations, a deadline shorter than the period is assigned and the scheduling criteria is the deadline. Deadline Monotonic Scheduling (DMS) assigns highest priority to the task with shortest deadline. As the schedule is built online, it is fundamental to know *a priori* if a given task set is schedulable. Two schedulability tests are provided: based on the CPU utilization rate and based on the response time.

- Based on CPU utilization rate.

  Liu and Layland [**?**] presented the first sufficient test:

  Theorem 2.2.1. *Any set of n periodic tasks is RM schedulable if the CPU utilization factor, U, is no greater than* $n(2^{1/n} - 1)$.

  The feasibility analysis of the RMS algorithm can also be performed using a different approach, called the Hyperbolic Bound, by Bini et al. [**?**]. It has the same complexity as the original Liu and Layland upper bound but it is less pessimistic, so allowing to accept task sets that would be rejected using the original approach.

  Theorem 2.2.2. *Any set of n periodic tasks is schedulable with the RMS algorithm if* $\prod_{i=1}^{n}(U_i + 1) \leq 2$*, being* U_i *the processor utilization factor per task.*

  The previous theorem also provides a sufficient condition for testing the schedulability of a task set under the RMS algorithm.

  In general, the deadline-monotonic scheme has not been employed because of the lack of adequate schedulability tests. RMS schedulability tests could be used by reducing the period of individual processes until equal to the deadline. Obviously such tests would not be optimal as the workload on the processor would be over-estimated [**?**].

  Tests based on response time provide precise results about the schedulability of task sets, both for RMS and for DMS.

- Based on response time.

  For arbitrary fixed priorities, including RMS, DMS, etc., the response time analysis allow to obtain an exact test (i.e., necessary and sufficient condition) in the following conditions: preemption, synchronous release, independent tasks and deadlines shorter than periods.

  This schedulability test is based on calculating the worst case response time (WCRT) [**?**],[**?**], i.e., the maximum time interval between arrival and finish instants for each task. If a task is activated at the same time as all other high-priority tasks, the WCRT will happen in the first activation of the task [**?**]. This is known as critical instant. Then, if, for each task, the WCRT is less than its deadline, the task set will be schedulable. The result of this test is not only whether the system is schedulable or not. It also provides the WCRT of each task and then, which tasks are involved in deadline misses, if in fact exist.

  The technique to calculate the WCRT is presented in equation 2.1:

$$WCRT_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{WCRT_i}{T_j} \right\rceil C_j \tag{2.1}$$

  The second term of the previous equation represents the interference suffered in a worst-case activation by all the *j* tasks with higher priority than the studied *i* task. Equation 2.1 is solved by iterative methods and stop conditions are the violation of a deadline ($WCRT_i \geq D_i$ for any task *i*) or the convergence ($WCRT(k+1) = WCRT(k)$).

In fixed priority based schedulers, as priorities do not change over time, the worst case response time is always held in the first activation, in contrast with what happens in dynamic priority based schedulers.

2.2.2.2 **Dynamic priority based schedulers**

In this scheme, the scheduler does not assign a initial priority to the tasks but at runtime. EDF (Earliest Deadline First) and LLF (Least Laxity First) are examples of optimal dynamic scheduling with dynamic priority. Dynamic priority algorithms

can be divided into two categories, depending on how priorities change while jobs are active. On the one hand, in job-level fixed-priority algorithms, jobs cannot change priorities. On the other hand, in job-level dynamic-priority algorithms, jobs may change priority during execution. For example, LLF [**?**] [**?**] algorithm is a job-level dynamic-priority algorithm. LLF assigns higher priority to jobs with smaller laxity, being laxity the time difference between the time until deadline and the remaining execution time. Since the laxity of a job changes over the time, the job priorities change dynamically.

However, EDF [**?**] is a job-level fixed-priority algorithm and is the most used algorithm. EDF assigns the highest priority to the task with the earliest absolute deadline. The EDF algorithm has been proven by Dertouzos [**?**] to be optimal among all scheduling algorithms on a uniprocessor, in the sense that if a real-time task set cannot be scheduled by EDF, then this task set cannot be scheduled by any algorithm. Due to the extensive use of EDF policy as dynamic priority based scheduler, EDF schedulability tests are now presented.

For tasks with deadlines equal to periods, Liu and Layland [**?**] presented a necessary and sufficient schedulability condition for EDF systems that satisfies the equation 2.2:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{2.2}$$

In these schedulers, the sufficient condition for schedulability is that the total processor utilization is less than 1. EDF requires more complex run-time system with higher overhead. Moreover, this test only ensures the schedulability (or non) of the system without offering any other information about tasks (for example, response times). EDF is unpredictable: domino effect may occur. It means that if a tasks misses its deadline, a large number of tasks could follow it.

For tasks with $Di \leq Ti$, schedulability analysis is more complicated and some definitions must be introduced:

Definition 2.2.1. [**?**] The function $G_\tau(t)$ represents the computation time demanded from initial time to time t for a set of tasks τ. It is calculated as:

$$G_\tau(t) = \sum_{i=1}^{n} C_i \left\lceil \frac{t}{T_i} \right\rceil \tag{2.3}$$

It is a positive and non-decreasing function that only increases when a task is released, that is, it grows as many units as time computation is required by the task that has been activated.

If tasks are simultaneously activated at time t=0 (i.e. ϕ_i=0 for all the tasks so the task set is synchronous), then:

Definition 2.2.2. [**?**] [**?**]The maximum cumulative execution time requested by jobs of a set of tasks τ whose absolute deadlines are less than equal to t is:

$$dbf_\tau(t) = \sum_{i=1}^{n} C_i \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \tag{2.4}$$

The *demand bound function* is a positive and increasing function that only increases in the so-called *scheduling points* i.e., when a deadline arrives.

Once these functions have been introduced, let us present different schedulability tests for EDF scheduling:

In 1980, Leung and Merrill [**?**] noted that a set of periodic tasks is schedulable if and only if all absolute deadlines in the interval $[0, max\{s_i\} + 2H]$ are met, where s_i is the start time of task τ_i and H is the least common multiple of the task periods. In 1990, Baruah et al. [**?**] [**?**] extended this condition for sporadic task systems and they showed that the task set is schedulable if and only if $dbf_\tau(t) \leq t \quad \forall t > 0$.

Studying the demand bound function over all time is a tedious process. For this reason, next schedulability tests consisted on the reduction of the time interval in which the previous schedulability condition must be satisfied.

Theorem 2.2.3. *[**?**] [**?**] [**?**] A general task set is schedulable if and only if* $U \leq 1$ *and*

$$dbf_\tau(t) \leq t, \quad \forall t \leq L_a$$

, where L_a *is defined as follows:*

$$L_a = max\left\{D_1, \cdots, D_n, \max_{1 \leq i \leq n}\{T_i - D_i\}\frac{U}{1-U}\right\} \tag{2.5}$$

In 1996, Ripoll et all [**?**] assume a different upper bound L_{a_2} for the schedulability test:

$$L_{a_2} = \frac{\sum_{i=1}^{n} (T_i - D_i) U_i}{1 - U} \tag{2.6}$$

, being $L_{a_2} \leq L_a$.

In 1996, Spuri [**?**] and Ripoll et al. [**?**] derived another upper bound for the time interval which guarantees the schedulability of the task set. This interval is called the *synchronous busy period*. It is a processor busy period in which all tasks are released simultaneously at the beginning of the processor busy period and ends by the first processor idle period. Its length is the maximum of any possible busy period in any schedule. The length of this interval is calculated by an iterative process [**?**] [**?**] Then, the schedulability condition is defined as:

Theorem 2.2.4. *[**?**] A general task set is schedulable if and only if* $U \leq 1$ *and*

$$dbf_\tau(t) \leq t, \quad \forall t \leq L_b$$

, where L_b *is the length of the synchronous busy period of the task set.*

As there is no relation between previous upper bounds, the time interval that has to be studied in order to ensure the schedulability of the task set is the minimum between L_a and L_b.

As presented before, research into uniprocessor real-time scheduling starts in the late 1960s. From then, different surveys as [**?**] and books as [**?**], [**?**] reviewing previous and other results have been presented. It might be considered that the uniprocessor real-time scheduling field has been studied in depth. However, there is still a big scope for future research in multiprocessor real-time scheduling, because it is a much more difficult problem than uniprocessor scheduling. The parallel execution on more than one core at a time has several implications on the software design and implementation, as scheduling and WCET calculus, core allocation and task migration, etc.

2.2.3 Multiprocessor scheduling

From 1960s, with the emergence of first multiprocessor scheduling theories [**?**] [**?**], until 2001, with the release of the first dual-core processor, POWER4, a dichotomy between task migration and fixed allocation of task to processors existed.

From 2001, advantages of increasing the number of processor cores where plausible and convincing in real-time field.

In a multiprocessor system, schedulability tests are based on the processor load of the overall system. The processor load in a time interval *t* depends on the demand bound function $dbf_\tau(t)$ (Equation 2.4) as follows [**?**]:

$$load(\tau) = \max_{\forall t} \left(\frac{dbf_\tau(t)}{t} \right) \tag{2.7}$$

And then, the necessary condition for a τ task set to be schedulable in a system with *m* processors is [**?**]:

$$load(\tau) \leq m \tag{2.8}$$

There are some considerations to be taken into account regarding to multiprocessor scheduling algorithms: priority and allocation problems [**?**].

Priority problem does not involve any completely new proposal with respect to scheduling techniques in uniprocessor systems, stated previously. Priorities could be fixed before the execution (fixed priority schedulers) or during execution time (dynamic priority schedulers).

Allocation is a new problem that appears with multiprocessors systems. It comes to answer the question: which processor will execute each task? This decision could be taken offline before the execution or online, at execution time. Allocation problem contemplates migration levels allowed in cores. Migration is defined as the fact of changing the task allocation between cores and, depending on the depth of the migration, is classified into:

- No migration. Once tasks are allocated into cores, they are executed always on that core. In these situations, the system is defined as fully partitioned system.

- Task-level migration. Each job of a single task could be executed in a different processor but, once it is released, it remains in that core until the end of its activation. This system is said to be semi partitioned system.

- Job-level migration. Task jobs can migrate and be executed in different processors. This system is referred to as global.

As partitioning is one of the main considerations of this work, let us develop in-depth this concept.

As stated before, fully partitioned systems are those in which migration is not allowed between cores. In these systems, once a task allocation to cores has been achieved, is possible to use uniprocessor scheduling algorithms in most of the multiprocessor systems.

In comparison to global scheduling, partitioned systems present some advantages that are defined as follows:

- There is no cost associated to tasks migration between cores since it is not allowed.
- As each processor uses a separate run-queue, its manipulation is simpler than a single run-queue.
- If a task exceeds its worst-case computation time, it will affect only to other tasks in the same processor and not to other processors. This fact makes possible the *isolation* previously stated.

The main disadvantage of the partitioning approach to multiprocessor scheduling is that the task allocation problem is analogous to the bin packing problem and is known to be NP-Hard [**?**].

(Generally speaking, partitioned systems refer to those systems in which migration is not allowed in an allocation problem. This work uses the concept of *partitioned* system referring to systems that encapsulate applications into partitions. From now on, partitioned systems concept will be used to refer to the latter.)

The classical bin packing problem is: given as a set of items with different size or weight, allocate them into as few unit sized bins as possible. In the context of this work, the items that have to be allocated correspond to partitions or tasks, the size or weight is the utilization of the task or partition and, finally, the bins are the cores where the items have to be allocated. Thus, bin packing determines the number of cores that will be used and on which processor each item is executed.

A number of heuristics are available for solving the bin packing problem. Some of the most well-known are:

- First Fit (FF). Each item is allocated into the first bin that it fits into, without exceeding the maximum capacity of the bin. If there is no one available, a new bin will opened.

- Best Fit (BF). This algorithm allocates each item into the most full bin where it fits, and, as FF, possibly opening a new bin if the item does not fit into any currently open bin.

  The first upper bound on FF and BF shown by Ullman in 1971 [**?**], which proved that FF, BF $\leq 1.7 \cdot$OPT $+3$, being FF or BF the number of cores used by First Fit or Best Fit respectively and OPT, the optimum solution. The absolute approximation of FF and BF was bounded by Simchi-Levy [**?**] and improved recently for FF only [**?**],[**?**].

- Worst Fit (WF). As opposed to BF, WF allocates each item into the bin that leaves more remaining capacity, i.e. the emptiest bin. It will also open a new bin if no one is available to allocate the item. In this case, the absolute bound is WF $\leq 1.7 \cdot$OPT [**?**]

In addition to these heuristics, there are other bin packing algorithms used to solve the allocation problem. Coffman et al. book [**?**] present a survey and classification of these algorithms.

Previous algorithms are very sensitive to the order of the items. For example, if items with small weight are allocated first, accommodating large items in the gaps they leave is a difficult work. For this reason, the first step in a bin packing algorithm consists of selecting which of the available items has to be allocated first through any of the previous algorithms. There are several methods to order the items before allocating them into cores. The most used technique consists of ordering the items according to their weight, i.e., the utilization. In this sense, random, decreasing and increasing utilization are the main variants. Decreasing utilization method (DU) puts the items in decreasing order by utilization. Increasing utilization method (IU) puts the items in increasing order by utilization. Random (R) orders the items arbitrarily.

After describing scheduling techniques in monoprocessor and multiprocessor systems, new trends used to meet requirements on time, power consumption, etc.,

are introduced. In next sections, several systems that moved away from the classical concept of real-time systems are included. Some of them are partitioned systems, hierarchical systems or mixed-criticality systems.

2.3 Partitioned systems

In the last years, modern computing systems have increased their processing capabilities and computational resources. In this way, they are able to execute several real-time applications in a mono-processor, sharing memory and other resources. From this point, the protection between applications in terms of execution time and memory space is needed. This gave rise to partitioned systems, developed to address security and safety problems. They have evolved to fulfil security and avionics requirements where predictability is extremely important[**?**]. The separation kernel proposed in [**?**] established a combination of hardware and software to allow multiple functions to be performed on a common set of physical resources without interference.

A partition consists of an encapsulated group of applications that provide independent execution on a common platform. The operating system is in charge of support the execution of the applications. Partitions are executed independently on the top of the hardware, which could be virtual or not.

Spatial and temporal isolation properties of the partitioned architectures are very relevant aspects in partitioned systems. Spacial isolation implies that each application can only access to its independent memory addresses. Temporal isolation means that only one application at a time has access to the system resources, making it impossible for an application to run when another application is running. Several projects ([**?**],[**?**],[**?**]) have been successfully developed using this approach in the avionic sector. Moreover, fault isolation is also considered. It implies that a fault in an application must be handled by itself or the system without affecting other applications.

From this concept, Integrated Modular Avionics (IMA) [**?**] is an architectural proposal that emerged to integrate several applications with different levels of criticality in a hardware platform. The European Space Agency (ESA) has promoted the adaptation of IMA for the new generation of satellites [**?**].

An IMA development process involves several roles like:

- System Architect (SA): The SA has responsibility to define the overall system requirements and the system design, including the optimal decomposi-

tion into hosted partitions jointly with the detailed resource allocation per partition.

- System Integrator (SI): The SI is responsible for verifying the feasibility of the system requirements defined by the SA, as well as responsible for the configuration and integration of all components.

- Application Suppliers (AS): An AS is responsible for the development of an application according to the overall requirements from the SA and the SI. AS shall verify that the allocated budget and safety parameters are respected. Assuming that each application is located in a partition and a partition can have only one application, an AS can also be called Partition Developer (PD).

There are other roles in the process but due to space restrictions we only detail those interesting for the purpose of the paper. For a complete description of the main roles and responsibilities see [**?**]

IMA made possible the definition of an architecture with new functionalities, as the parallel execution of applications and fault handle at different levels. Recently, the ESA presented the Time and Space Partitioning program (TSP) that provides an execution environment in which software components are executed without interference. Its benefits are related to the allocation of different criticality partitions that coexist within the same computer, management of the growth of software functionality, achieve higher degree of integration as more performing processors becomes available and facilitate design for re-use [**?**], [**?**].

In Figure 2.3, the general scenario of this work is depicted. It is a multi processor system based on hypervisor. Applications are encapsulated in partitions with different levels of *importance* and are allocated in cores following any scheduling algorithms.

All the elements shown in Figure 2.3 play a key part and are highly relevant to This thesis:

- Multicore system: using different cores for enhance performance, reduce power consumption and obtain more efficient simultaneous processing of multiple tasks.

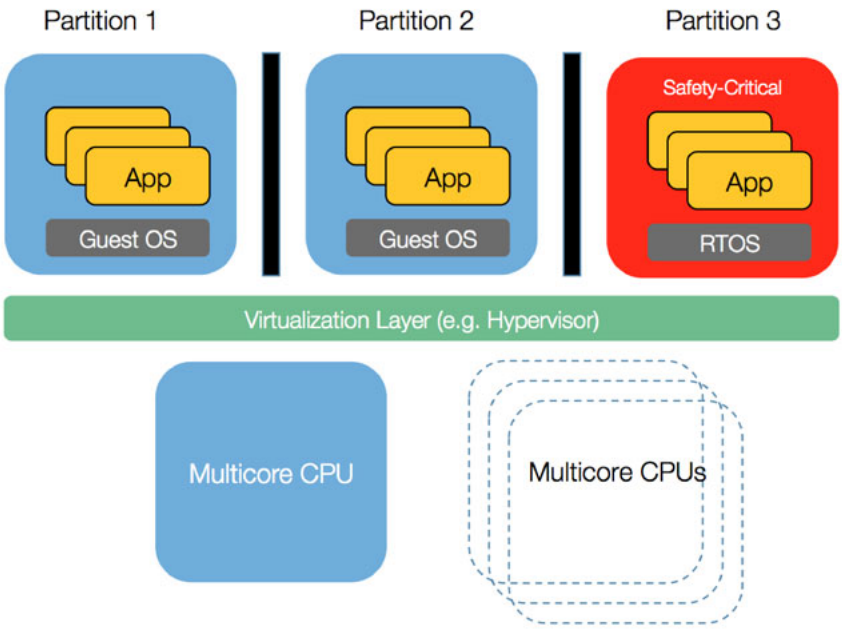


Figure 2.3: Virtualized OS architecture on a multicore processor

- Virtualization layer, for example, provided by a hypervisor. More detailed information about virtualization techniques in real-time systems is developed in Section 2.7.
- Partition: unit allocated to cores in order to be executed. If a system is composed of partitions with different *importance* levels, i.e. criticality, it will be a mixed-criticality system (MCS). See Section 2.5 for detailed information.
- Application: each partition consists on one or a group of applications or processes, that are groups of tasks that contain threads of execution.

Once all the elements of a virtualized multicore partitioned real-time system are defined, the concept of hierarchical scheduling is introduced.

2.4 Hierarchical scheduling

In order to schedule partitioned systems, hierarchical scheduling techniques must be introduced. Based on the proposed software architecture in an IMA system (stated in chapter 2.3) where a hypervisor supports the execution of several temporal and spatial isolated partitions, the system can be modelled as a hierarchical real-time system in which tasks are allocated to partitions.

Partitioned software architectures define two layers: the global layer, with a scheduler that allocates CPU time to partitions, and a local scheduler per partition, which schedules the tasks using the available time per partition.

For local and global scheduling, there are several techniques in order to schedule partitioned systems: cyclic scheduling, fixed or dynamic priority schemes, etc. As previously emphasized, meeting deadlines is a key aspect in real-time systems. In recent years, some works have dedicated effort to the calculation of the exact worst-case response time (for example [**?**]and [**?**] in fixed-priority scheduling).

The strategies that can be followed to achieve hierarchical scheduling are [**?**]:

- Server-based scheduling. Schedulers allocate an executing capacity and replenishment period to each partition. In this way, a separate server is allocated to each partition, so that each server will never exceed a predefined bandwidth. The main disadvantage of this approach is the difficulty to undertake complex task models and systems with high levels of criticality [**?**].

- Compositional scheduling. Each partition requests an amount of computation and the scheduler tries to satisfy all partition requirements. In this way, all partitions will require their pessimistic amount of computation time to assure schedulability and, for this reason, this strategy is less efficient than others. However, predictability is the most possible aspect, specially with cyclic executives of partitions and also in large complex systems. This approach makes possible the isolation between partition developers (PD) and system integrator (SI), roles defined in section 2.3. PD do not provide information about the schedulability methodology in the applications level, but only provide the temporal abstract interface of the partition [**?**].

- Flat scheduling. The interrelation between tasks is analysed previously, considering all of them as a global system. Then, tasks are allocated into partitions with the purpose of reducing the number of context switches, grouping or trying to group them. With this, the final schedule will be very efficient and, sometimes, optimal. In this strategy, the timing knowledge of tasks should be previously detailed in depth in order to analyse the overall system and optimize the solution. Moreover, changes in any task suppose a re-study of the whole schedule. This fact and the no isolation between PD and SI are the main disadvantages of this approach.

As hierarchical scheduling allows temporal and spatial isolation between partitions, this independence enables each partition to have a different criticality level, depending on the consequences that failures cause in the system. The concept of mixed criticality system is introduced in next section.

2.5 Mixed criticality systems

In a real-time and embedded system, the execution of processes with different importance (hereafter, criticality) in the same platform is called mixed criticality system (MCS). This is a current and important trend in systems like avionics, space, etc. In these systems, high criticality applications are the most costly to design and verify. These kind of systems present advantages in terms of cost, space, weight, heat generation and power consumption. An example of these applications is an aircraft: in-flight information system has much lower criticality than flight control systems and both coexist in one “mixed criticality" machine.

A MCS has two or more distinct levels (for example safety critical, mission critical and low-critical). The level of criticality of each application, L_i, is set at design time by the system engineers responsible for the entire system. Vestal [**?**] presented in 2007 the first seminal paper on the verification of a MCS. In this paper, different WCET per task are considered, depending on its criticality. It means that, the higher the criticality level of the tasks, the bigger WCET will be needed to ensure safety of the system. For example, if a task is defined as safety critical, its WCET will be higher than if it is considered mission critical or non-critical. These changes in temporal parameters of the system make necessary a reconsideration of the schedulability of the overall system, in order to guarantee its feasibility. Then, in MCS a task is defined by $\left(\overrightarrow{T}, D, \overrightarrow{C}, L\right)$, being $\overrightarrow{T}$ and $\overrightarrow{C}$ vectors of values according to the criticality level of the task. As stated before, if L_1 and L_2 are two criticality levels and $L_1 > L_2$, then $C(L_1) \geq C(L_2)$ and $T(L_1) \leq T(L_2)$. Another point to consider is the criticality execution mode of the system. At the beginning, the system runs in the lowest criticality level. If any task or activation requires more time and deadlines are jeopardized, the system mode changes to a high level of criticality. Then, no tasks with low criticality level will be executed from this time and the system remains in the highest criticality level.

In general, almost all papers consider two criticality levels, HI and LO, being HI > LO. Then, a task *i* is defined as $(T_i, D_i, C_i\,(HI)\,, C_i\,(LO)\,, L_i)$, with $C_i(HI) \geq C_i(LO)$. Most of later works made use of this consideration and adopted the “Vestal model" o short variations. There are other works that propose other models as [**?**].

Vestal seminal paper showed that neither RMS nor DMS priority assignment was optimal for mixed-criticality systems. Baruah and Vestal [**?**] generalised Vestal's model by using a sporadic task model and by assessing fixed job-priority scheduling and dynamic priority scheduling. It contains the important result that EDF does not dominate FPS when criticality levels are introduced, and that there are feasible systems that cannot be scheduled by EDF.

The first paper that considers multi-processor or multi-core platforms in MCS was [**?**]. These authors implemented an scheme called MC_2, used for different authors to evaluate different considerations as OS overheads [**?**], isolation techniques [**?**], parallel tasks [**?**], etc. [**?**] also introduced multi-processor issues and virtualization in MCS.

Contrary to Vestal model with different levels of assurance for WCET (HI and LO), other recent approaches use a model based on probabilistic WCET (pWCET) distributions. The pWCET is a probabilistic distribution which upper bounds all the possible execution times of a task [**?**] and each of these values is associated with the worst case probability of being exceeded. Although probabilistic models present less pessimism than deterministic models, the complexity cost of probabilistic distribution increases. Some works extend the Vestal model to the probabilistic case, such as [**?**] [**?**]. The results of probabilistic schedulability tests consist of probabilistic worst case response times (pWCRT). These probabilistic models calculate the increasing resource demands at run-time and take decisions about pWCRT in the system scheduling [**?**]. As mentioned in previous sections, to ensure TSP, this work does not consider online decisions neither unpredictable or undesired situations, so indeterministic approaches as probabilistic distributions are not desired.

Next sections show how Vestal model does not completely meet the requirements of the project in which this thesis is based on. Moreover, other theories on how impractical this model is are also discussed. Although the most relevant methods for multiprocessor partitioned real-time systems have been presented, there are some challenges that remain unsolved, specially with the appearance of battery-operated embedded devices. In this kind of systems, energy saving is a vital issue for developers and, more particularly for users. Reducing heat dissipation involves

in increasing the long-term availability and reducing cooling equipment costs. Energy savings aspects are considered bellow.

2.6 Energy savings policies

Energy management is a very active research are in the recent years. In fact, effective energy management is essential for those systems which are portable embedded and require energy autonomy. Autonomous mobile robots, industrial controllers, wearable devices, mobile phones, etc. are examples of battery-powered embedded systems. The battery lifetime is one of the most important characteristics of a portable device. As sometimes replacing batteries is not possible, minimizing the energy consumption is a key procedure to provide a longer lifetime. Energy minimization is one of the main requirements during the design phase of embedded systems for several reasons: cost reduction, thermal issues, etc. Power density of modern electronic circuits is increasing and this may cause failures [**?**].

In real-time embedded systems, two widely used techniques for reducing the energy consumption are *Dynamic Voltage and Frequency Scaling* (DVFS) and *Dynamic Power Management* (DPM). On the one hand, DVFS is based on adjusting the CPU voltage and frequency on-the-fly in order to reduce the overall energy consumption [**?**]. Since reducing the system frequency increases the task execution times, to guarantee the timing constraints when DVFS is applied, is extremely important for real-time systems. On the other hand, DPM is based on switching the processor to a low-power inactive state as long as possible, under the requirement of accomplishing with the real-time constraints . From this point of view, the problem to be solved is: *minimize the system energy consumption applying DVFS or/and DPM while satisfying temporal constraints of the tasks*. Lot of works about this problem have been published and [**?**] and [**?**] are excellent surveys of design techniques for energy-aware purposes in real-time safety-critical systems.

Recently, [**?**] presented a survey of energy management techniques for embedded systems.

Another survey of energy-aware scheduling algorithms for real-time systems is [**?**]. It presents a classification of the existing approaches for uniprocessor and multiprocessor systems based on DVFS, DPM or both.

Before presenting a overview of the energy-aware algorithms, the most relevant power models used in literature for energy-aware purposes are introduced.

The power consumption function, $P(f)$, as a function of the processor frequency f is generally formulated as follows [**?**]:

$$P(f) = K_3 f^3 + K_2 f^2 + K_1 f + K_0 \tag{2.9}$$

, where K_3 refers to the power of components that vary with frequency and voltage, K_2 refers to the non-linearity of DC-DC regulators in the range of the output voltage, K_1 represents the subsystems that operate at a fixed voltage but frequency can vary and K_0 represents subsystems than operate at a fixed voltage and frequency, i.e., are not affected by the processor speed.

One of the most common ways of expressing the power function (for example [**?**], [**?**] and [**?**]) is

$$P(f) = P_{sta} + P_{dyn} \tag{2.10}$$

In previous equation, P_{sta} stands for the static power consumption due to the leakage current and is independent of the system speed. P_{dyn} represents the dynamic power consumption due to switching activities and is assumed to be a polynomial function of the frequency *f*. This polynomial function is assumed to be defined as $P_{dyn} = \beta \cdot f^{\alpha}$, where α is a fixed parameter determined by the hardware and is assumed to be $2 \leq \alpha \leq 3$ [**?**] [**?**]. β is a constant that depends on the effective switching capacity and is assumed to be $\beta > 0$.

During an interval Δt, a processor runs at a frequency f and executes Δc cycles so that $\Delta t = \frac{\Delta c}{f}$. As power is how quickly energy is used or transferred, the energy function can be expressed as:

$$E(f) = (P_{sta} + \beta f^{\alpha}) \frac{\Delta c}{f} \tag{2.11}$$

The lowest available frequency that minimizes the energy consumption of the system is equivalent to the first order derivative of Equation 2.11 equal to zero. Then, the minimum value for $E(f)$ happens when:

$$f_{min} = \sqrt[\alpha]{\frac{P_{sta}}{\beta\,(\alpha - 1)}} \tag{2.12}$$

f_{min} is denoted as *critical frequency* and represents a frequency below which it is not beneficial to reduce frequency energy-wise. This is shown in some works as [**?**] and [**?**] and an example can be found in [**?**].

Reducing the voltage and the frequency of the processor supposes an increase in the execution time of the tasks allocated in it. In these situations, a control over accomplishing temporal requirements should be taken into account, specially in real-time systems. In the major part of works, the *worst-case execution time (WCET)* $C_i(f)$ of a task τ_i is considered to be fully scalable with the inverse of the frequency [**?**], [**?**], i.e., $C_i(f) = \frac{C_i}{f}$. However, some works claim that this is only an upper bound [**?**], [**?**] because there is a fixed portion of the WCET not affected by speed changes. This happens often with some operations on devices that do not share the clock frequency with CPU, such as input/output operations. Thus, another way to represent the computation time of a task is as a sum of a variable and fixed part: $C_i(f) = C_i^{fixed} + \frac{C_i^{var}}{f}$.

Once the models have been analysed, following is a summary of the taxonomy used to organize the energy-aware algorithms previously defined, DVFS and DPM, in platforms with single-core and multicore CPU. Figure 2.4 represents the taxonomy in the single-core scheme.

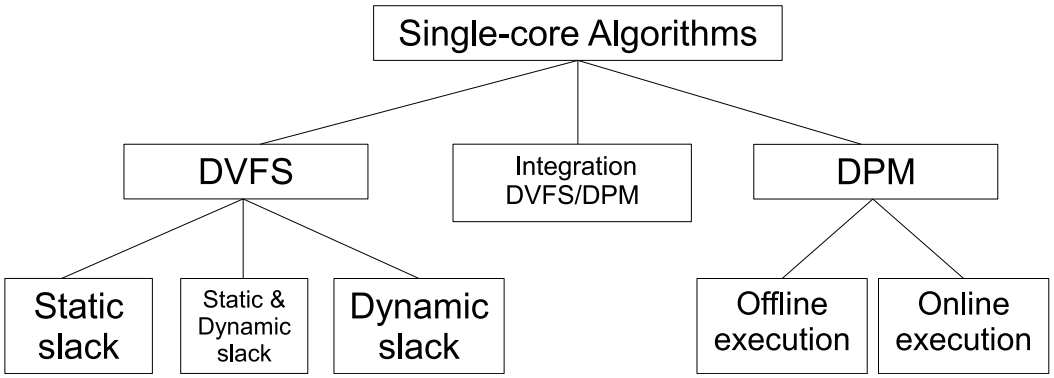


Figure 2.4: Taxonomy of energy-aware algorithms for single-core systems.

DVFS algorithms adjust the system frequency or speed according to the unused CPU time (*slack*) in three ways: statically, dynamically or both. On the one hand, static DVFS benefits from the residual processor utilization taking into account the worst-case execution time of the tasks. On the other hand, dynamic DVFS considers the difference between the worst-case execution time of the tasks and their actual execution time.

DPM algorithms are classified depending on when the decisions about the adjust of frequency system are taken: offline or online.

Integrated algorithms use DVFS and DPM techniques in order to minimize the energy consumption of the system.

Among hundreds of works related to these techniques, different considerations have been taken: to account or not the time overhead due to changes in the speed, to consider continuous or discrete frequencies, to fully scale the computation time with the frequency or only partially, etc. Considerations that concern to this work will be detailed in next sections.

The taxonomy used to classify the multicore algorithms is depicted in Figure 2.5. The main branching separates those platforms which share the clock between a subset of cores (*voltage island*) from those which allow different frequency for each core (*independent frequency*). In the latter case, the distinction between the adjust of frequencies per core or per task is made.

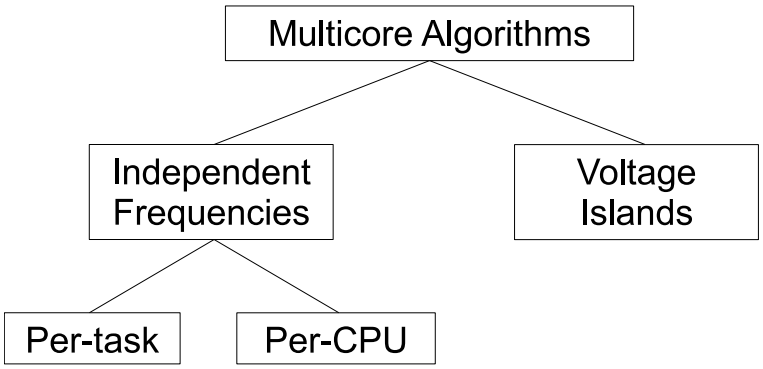


Figure 2.5: Taxonomy of energy-aware algorithms for multicore systems.

In our particular case, this work falls under the scope of multicore algorithms. This is the reason that the attention is turned on this field. Since 2003, resource management for multiprocessors has been an active research area in real-time systems. Then, designers realised that increasing the system frequency in order to increase the overall performance was not suitable in terms of energy consumption and the existing studies about energy-aware techniques had to be extended to multiprocessors.

When classifying energy-aware algorithms, the main distinction is based on the DVFS support provided by the system. If a different frequency for each core is

allowed, algorithms do form part of *Independent Frequencies* group. However, if frequency is shared among all cores, they are classified as *Voltage islands*.

In those systems in which frequency and voltage of each processor is adjusted independently, algorithms are classified into two categories:

- per-task DVFS: the frequency is fixed per task, as in sigle-core algorithms. First work [**?**] addressed only dynamic power and was applied to systems with typical specifications by the time that paper was written. However, authors noticed that any processor could be switched to a low-power inactive state, so an integration between DVFS and DPM was proposed. Same authors proposed later an extension of the previous approach but considering precedence constraints between tasks. Then other algorithms were proposed: [**?**] considered the division between tasks, which is very useful for tasks with high utilization factors; [**?**] proposed an energy-aware algorithm that applies to parallel tasks and [**?**] presented an approach based on mixed integer linear programming to optimize DVFS and DPM at the same time.

- per-CPU DVFS: the frequency is fixed per core, independently of the tasks allocated to that core. Different works have been published in this area but all of them consider periodic task sets with implicit deadlines. For example, [**?**] considers only dynamic power consumption without taking into account switching overheads. One of the main contributions of this work (and it will be considered also in this work) is that the most balanced core is the most energy-efficient system. Other approaches with different considerations have been published since 2003 [**?**] , [**?**], [**?**] but it took until 2012 with [**?**] to find an optimal solution. This proposal considers uniform processors with different discrete frequencies per core and periodic tasks sets. Since this is an offline algorithm, overhead due to the change of frequency is avoided and static power is not considered.This approach consists of setting frequencies at the minimum value and proceed to increase the frequencies per task until finding a feasible allocation of frequencies to tasks for energy-aware purposes.

Those systems in which frequency and voltage are shared among cores are defined as voltage islands. This approach provides a solution to both problems: the

hardware complexity when providing different DVFS per core and also the impossibility of energy savings when all cores share frequencies. Since 2010, different energy-aware algorithms have been presented. [**?**] proposed a complete approach based on multiple sleep states and frequencies shared among cores. This algorithm computes offline the optimal numbers of cores and allocates tasks in the cores. At runtime, the working frequency and the number of idle cores could be characterised. Other works as [**?**], [**?**] and [**?**] considering continuous or discrete speed values, different power model and complexity of the algorithms are proposed.

In spite of the extensive literature previously detailed, applying previous techniques for mixed criticality systems is no easy task. Until now, only some works have been published with the view to resolving this problem. The first paper that considers the importance of energy in mixed criticality systems is [**?**], which defined energy as important as timing in terms of compliance of mixed-criticality guarantees. [**?**] presents a DVFS method to minimize energy for unicore mixed-criticality systems. [**?**] proposes another method with the same purpose but only considers dynamic power consumption and forgets static energy consumption. Other works reduce or even sacrifice the performance of low criticality tasks in order to reduce the energy consumption of the system. Some examples are [**?**], which is based on DPM technique, and [**?**], which tries to improve the energy utilization on critical tasks. Works that consider allocation techniques and schedulability in mixed criticality systems are [**?**] and [**?**]. Both assume the Vestal model [**?**] for MCS and aim at enhancing system schedulability instead of saving energy. However, [**?**] proposes a DVFS technique for energy-aware purposes for MCS on identical multi-core processor and also considers the schedulability of the overall system.

One work that focuses on energy efficient allocation in MCS is presented by Awan et al. in [**?**]. They present an approach to generate a set of feasible allocations and select the one with the lowest energy consumption. They also assume the Vestal model for MCS. They also do not take into account the different frequencies of the cores in the allocation algorithm.

In [**?**], authors also propose an energy saving method for MCS on multicore based on the isolation of tasks depending on their criticality levels on different

cores. In our study, we have consider the mix of workloads from different criticality levels in different cores.

As seen in dates of the previous publications, energy consumption minimization in mixed-criticality systems is an open topic that attracts interest at present.

2.7 Virtualization

In order to reach TSP, different techniques as separation kernel or virtualization are proposed. While a separation kernel requires the use of the same operating system, virtualization allows a different operating system for each partition (real-time operating system, general purpose operating system or bare-metal applications).The concept of virtualized open source platforms for mixed criticality embedded systems is dealt in this section.

Virtual machine technology can be considered the most secure and efficient way to build partitioned systems. They use virtualization techniques in order to achieve the desired temporal, spatial and fault isolation previously mentioned. A hypervisor or virtual machine monitor (VMM) is a layer of software (or a mixture of hardware and software layer) that runs several partitions in a single computer. The main difference between hypervisors and other kind of virtualizations is the performance.

The main characteristics of hypervisors are low overhead and a performance similar that it would have if it may be executed in the native system.

When a hypervisor is designed for real-time embedded systems, the main issues that have to be considered are: temporal and spatial isolation, basic resource virtualization (clock and timers, interrupts, memory, CPU time, serial I/O), real-time scheduling policy, deterministic hypervisor system calls, efficient inter-partition communication, efficient context switch, low overhead and low footprint [**?**].

Nowadays, hypervisors integrate ARINC-653 or AUTOSAR standards in their designs in order to be adapted to different domains. Several hypervisors for real-time in safety-critical systems have been developed. [**?**] is a survey that details some of them. For example, VxWorks 653 and VxWorks MILS are provided by WindRiver, LynxSecure provides an hypervisor for MILS and ARINC-653, RT-Xen is an extension of Xen that supports virtual machines with real-time performance requirements, Xtratum is a hypervisor that can be used to build a MILS architecture, etc.

Hypervisors are classified in two main categories [**?**]: native hypervisors and hosted hypervisors. Bare metal or native hypervisors (type 1) run directly on the

platform hardware, providing all the features needed by the guests. Hosted hypervisors (type 2) run on top of an existing OS and leverage the features of the underlying OS. Bare metal hypervisors include the early mainframe hypervisors as well as VMWare ESX server, and kvm, a hypervisor for Linux. Hosted hypervisors include VMWare GSX server.

Among these kinds of hypervisors [**?**], bare-metal or type 1 hypervisors are recommended for critical real-time systems because they provide better performance and control than hosted hypervisors. In the context of spatial and avionics systems, XtratuM is an hypervisor designed initially to replace RT-Linux with the purpose of meeting safety critical real-time requirements [**?**],[**?**],[**?**]. It is a type 1 hypervisor that uses para-virtualization, a virtualization technique in which the conflicting instructions are explicitly replaced by functions provided by the hypervisor and whose operations are as close to the hardware as possible.

XtratuM is adapted to deal with heterogeneous multicore architectures, in the framework of the MultiPARTES project [**?**].

From its version 1.0, Xtratum has been adapted to multiple platforms and, nowadays, is designed for as SPARCv8 (LEON 2/LEON 3/LEON 4), ARMv7 (Cortex R4/Cortex R5/Cortex A9), X86 and PowerPC architectures, especially in the framework of the MultiPARTES project [**?**]. Figure 2.6 depicts and overview of Xtratum architecture.

XtratuM main features are:

- Implementation of a cyclic scheduling policy, through a static scheduling plan that allocates time to partitions. Each partition internally schedules its own tasks following an algorithm. This cyclic scheduler follows ARINC-653 standard and ensures strong temporal isolation.

- Spatial isolation, all partitions are executed in processor user mode, and do not share memory.

- All the information of hardware resources, virtualized devices configuration and scheduling plan is defined by means of a configuration file.

- Provides robust communication mechanisms, based on ARINC-653 queuing and sampling ports. This information is also included in Xtratum configuration file.
- Provides an advanced health monitoring and error reporting, in order to detect and manage unexpected events. By the time an error is detected, some preconfigured actions to minimize its impact on the system are executed.
- Powerful configuration and validation tool (Xoncrete).

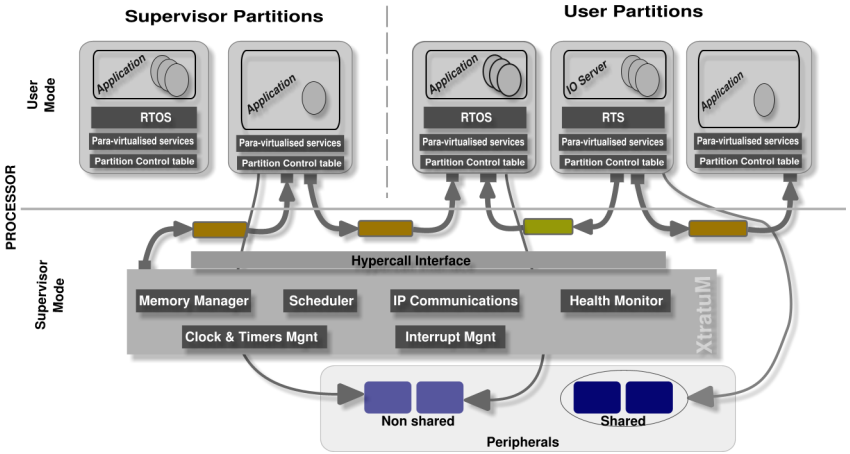


Figure 2.6: Xtratum architecture.

Xtratum is involved in near future space missions focused on different areas: measurements of CH_4, detection of gamma-ray bursts of the massive stars explosions, neutron stars mergers or black holes, measurements of surface water height of lakes, rivers and flood zones, and of deep and coastal oceans or three years European Space Agency mission, launched in 2022, and reaching Jupiter in 2030.

Due to the importance of all this missions and the impact they may have into the environment, a certification process in order to validate the product and ensure its reliability. Certification aspects are addressed in the next section.

2.8 Certification aspects in critical real-time systems

Certification process is a crucial aspect in fields as computing and software development in many safety-critical embedded systems. Specially, in these systems with strict timing and response requirements, product and process certification are the most challenging in developing software in this field. As stated before, failures in safety critical real-time systems may involve catastrophic damages and this fact makes necessary to stablish standards and guidelines for the developers to regulate critical processes. Thus, certification process guarantees the proper operation of the certified system or product, assuring that technical requirements are satisfied and injuries should not be caused.

As already mentioned, the development of safety-critical real-time systems requires the use of standards. The certification of these applications is based on very conservative assumptions and they usually exceed what is required by the designer's assurance levels [**?**]. A large number of standards and reports related to certification, validation, etc. have been issued, depending on the regulated industry: aerospace, avionics, railways, etc. The entity that issues digital certificates is called Certificate Authority (CA). Before the systems starts, the pessimistic assumptions of the CAs as well as design requirements have to be certified.

Figure 2.7 shows the process of integrating two applications with two different criticality applications in the same computing platform. While safety critical applications require complex calculus of BCET and WCET, mission critical applications have soft deadlines based on Quality of Service metrics [**?**]. However, the overall system has to meet the most strict requirements in order to be certificated.

Partitioned systems suppose a way of integrating applications with different criticality levels in multi-core and virtualized architectures. Applying this technology makes necessary the reconsideration of the classical certification criteria, based on the re-study of the overall system for coping with changes. This means that the certification cost and complexity can be reduced using partitioned approach, due to the modularity it offers. In this sense, thanks to partitioning, each partition is certified to the corresponding level of criticality [**?**]. However, providing sufficient assurance of the veracity of this method is a technical challenge [**?**].

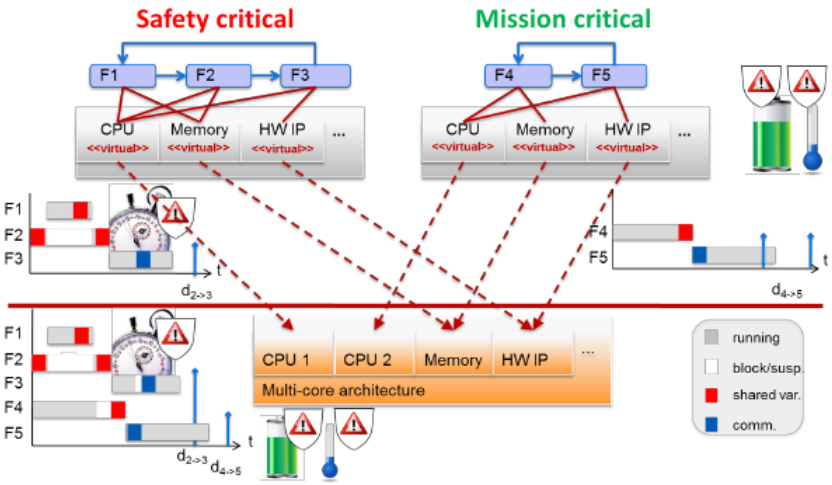


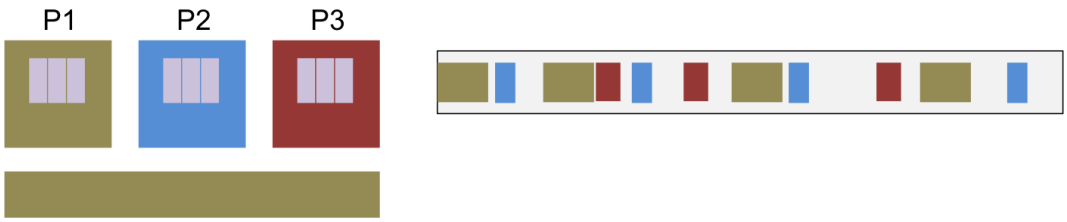
Figure 2.7: Integration of mixed critical applications on single chip.

IEC 61508 [**?**] is a generic international safety standard that provides a baseline for other specific standards applied to different domains: machinery, automotive, etc. Generally speaking, it defines the requirements needed by devices, systems, etc. in order to assure their safety functions while reducing risks.

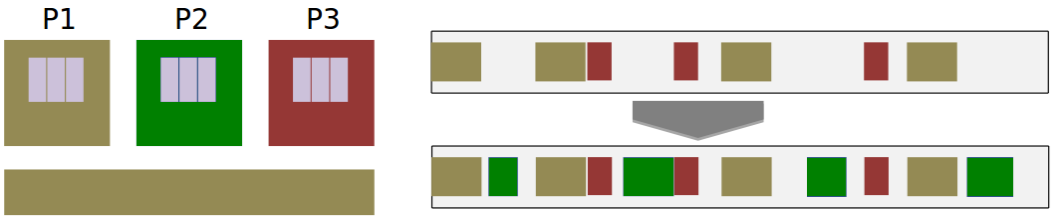
Safety standards as IEC 61508 state that "*whenever a system integrates safety functions of different criticality, sufficient independence of implementation must be shown among these functions."* [**?**] This sufficient independence happens when the probability of a dependent failure between high and low critical parts is sufficiently low in comparison with the highest safety integrity level. For this reason, spatial and temporal isolation is crucial to ensure the independence of execution. Different safety standards for different domains and needs exist, as DO-178/ED-12 with temporal and spatial requirements for avionics domain.

In the practice of our context, in a partitioned system, to achieve an independent certification of partitions offers interesting characteristics. For example, a partitioned system with 3 partitions and its possible scheduling plan is shown in Figure 2.8(a). Let us imagine that temporal requirements of partition 2 (in blue) change (in green). Then, it is unnecessary to repeat the certification process of the overall system but only in the partition 2.

The same happens in a system in which a new partition is added in order to use



(a) Initial situation



(b) One of the partitions changes its temporal requirements

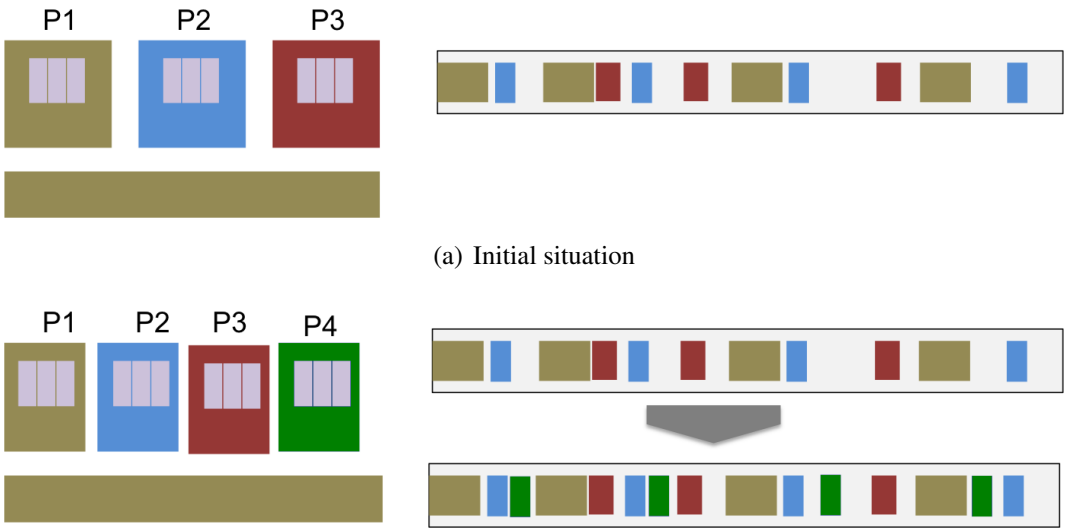
Figure 2.8: Changing temporal parameters in one partition.

the idle time of the system (2.9).

As seen before, it is not necessary to re-certificate the overall system when changes occur in any of the partitions. Thus, the following aspects may be considered in order to achieve an independent certification per partition:

- Temporal and spatial isolation between partitions must be achieved.
- Each partition requires a secure independent method of analysis.
- The scheduling of other partitions should not interfere in the studied partition.
- WCET analysis techniques should model the possible interference of multicore execution.

In order to achieve previous constraints, some considerations have to be addressed with respect to the scheduling policy of the virtualization layer. One solution that prevents the variability of the partition execution is the use of the cyclic scheduler, which also preserves the bandwidth of the processor. This method is proposed by the standard ARINC-653 [**?**] and previously defined in Section 2.2.



(a) Initial situation

(b) Addition of a new partition

Figure 2.9: Adding partitions to the partitioned system.

In this section, general aspects about certification process have been presented. Although there are many other important points, only the relevant items for this work have been provided.

2.9 Conclusions

According to the works reviewed in this chapter, there are several research challenges regarding to scheduling in mixed-criticality partitioned and hierarchical real-time systems, specially in multicore systems. Although single-core scheduling could be considered as deeply studied field, multicore scheduling, and specially, applied to those systems in which applications with different levels of criticality exist, still have a long way to go.

Furthermore, with the growth of battery-operated devices, it is important to have a complete control about energy consumption issues in these systems. This topic has been of enormous significance in recent years, in which portable devices are part of daily life. For this reason, energy-aware researches have considerably increased and the most relevant works have been covered in this chapter.

Although the literature presented in this chapter covers multicore scheduling and energy-aware aspects, there are some remaining challenges that this thesis solves. With regard to scheduling, this work aims at achieving a scheduling approach for hierarchical systems with unknown scheduling policies in the global level, as outlined in Section 1. To our knowledge, all existing works consider known scheduling policies in both, local and global level. Therefore, next chapter introduces a proposal of generation of offline scheduling plans in real-time partitioned field, using from the current literature as a starting point. With regard to energy saving aspects, only few works integrate MCS and energy savings concepts and most of them rely in classic models that do not completely fit our requirements.

Therefore, following chapters address scheduling approaches to overcome all challenges outlined in previous sections. Each section presents both non covered aspects in current literature and those that do not conform completely our requirements and also the new proposals to cope with them.

CHAPTER

3

Generation of offline plans in real-time partitioned systems.

Meeting deadlines of all tasks is clearly a crucial aspect during the execution of real-time systems. For this reason, a schedulability analysis is always necessary in this kind of systems. Particularly in certain situations, a partitioned and hierarchical system needs an offline schedulability analysis. In spite that lots of schedulability algorithms exist, to the best of our knowledge, they do not consider arbitrary policy in the global level. This chapter copes with schedulability of a hierarchical system, when the global level policy is not defined by any known policy.

The outline of this chapter is established as depicted in Figure 3.1.

3. GENERATION OF OFFLINE PLANS IN REAL-TIME PARTITIONED SYSTEMS.

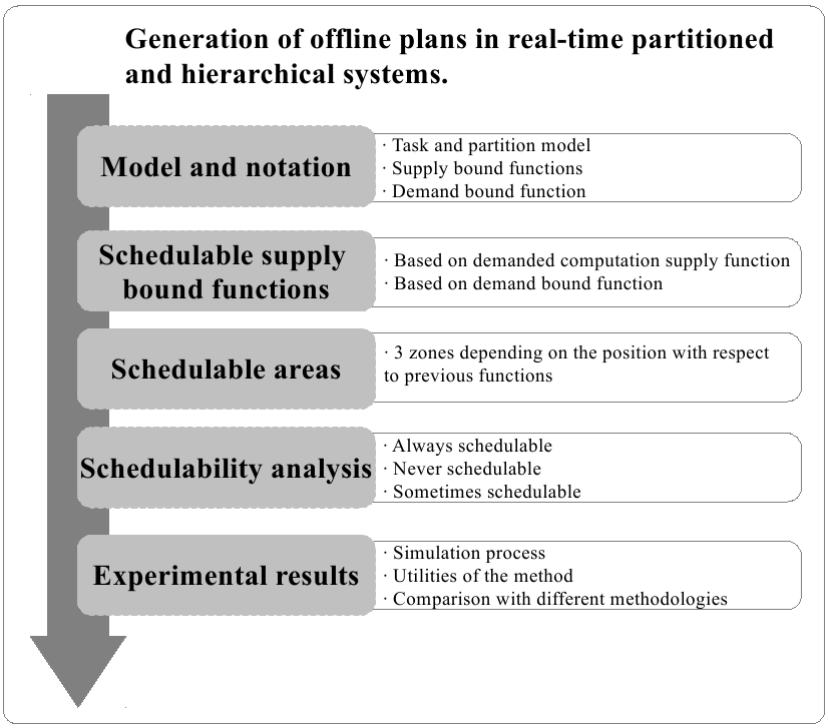


Figure 3.1: Chapter 3 layout

3.1 Introduction and objectives

As previously stated, in an IMA system, the SI is responsible for allocating execution time to applications. Furthermore, the PD manages this time internally for each application. In this sense, in a partitioned and hierarchical system, the SI allocates CPU in the global level, with a global scheduling policy, and the PD schedules tasks in the available CPU time through a local scheduling policy. In this sense, the SI ensures the feasibility in the global level and PD, in the local level.

In an IMA system where a hypervisor supports the execution of several temporal and spatial partitions, the configuration file of the system defines a temporal offline plan to schedule partitions in the global level. Throughout a hyper-period, partitions have different slots of time assigned to the execution of their tasks. For example, as seen in Figure 3.2, the SI assigns CPU periodically to the PD (depicted as time slots in black) and, locally, the PD schedules the tasks following any scheduling policy. As it is obvious, the execution of the tasks in the local level can not exceed the available time supplied by the SI. The PD requires the SI its temporal specifications, usually in the form of CPU bandwidth. Separately, the SI calculates and assigns the bandwidth given to the partitions using static cyclic executive scheduler.

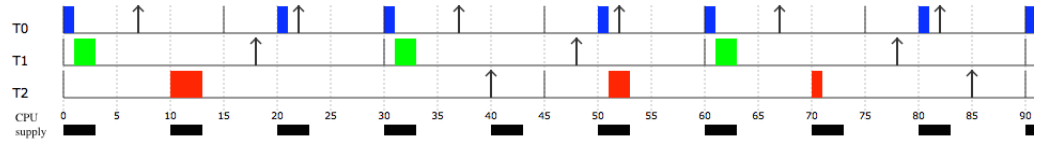


Figure 3.2: Execution chronogram and CPU supply of a partition.

If the assignment is made using a periodic resource or a bandwidth algorithm, the corresponding schedulability tests are available in the literature. All schedulability tests for the most well-known techniques were presented in Section 2. However, if this assignment does not follow any existing schedulability algorithm, no existing works that solve this problem are found in literature. Section 1.1 shows some examples that motivate the arbitrary assignment of time slots to a partition. Some of these examples were the addition of a new partition using the idle time of the current system or changing the temporal requirements of any partition already scheduled. Both situations require using the idle time in order to introduce

the new temporal load and this idle time does not follow any known allocation. As the schedulability tests for hierarchical systems are based on the calculation of the worst case response time in both levels, the overhead due to the global level can not be calculated since the global scheduling policy is not known. As a consequence of this problem, this work provides a proposal in order to analyse the schedulability of a task set in the local level of a hierarchical system, in whose global level the scheduling policy is arbitrary.

In this chapter, the problem that is being studied falls within the scope of the schedulability of a hierarchical system composed of two levels: local and global level. The global level policy does not follow any known policy but it is arbitrary. In other words, the SI assigns to the PD a sequence of time slots not derived from any known scheduling algorithm. Of course, this proposal is also valid if the scheduling policy is known in the global level.

In this section, two schedulable arbitrary slots assignments are presented. Both assignments or, from now on, supply bound functions, ensure the schedulability and feasibility of the task sets. Moreover, a schedulability analysis of a task set is presented, relating any arbitrary CPU supply with the functions previously presented.

In order to develop the calculation of these supply bound functions, a simulator that generates and studies these functions is implemented. Finally, a comparison between our method and other similar works is proposed, since our method can also be applied with known schedulability policies.

3.2 Model and notation

Our model is concerned with the pre-emptive scheduling of real-time applications on a uniprocessor. Each application consists of a number of *partitions* $P_1, .., P_m$. Each partition comprises a number of tasks. Thus, our hierarchical system has two levels, the partition (or global) level and the task (or local) level, each of them with its own scheduling policy. In this work, we will assume that the local level is scheduled under EDF scheduling policy and the global level is scheduled under any scheduler. The information regarding the global scheduling is provided as temporal windows or slots in which a partition is allowed to execute.

From now on, the sub index used to refer to a partition will be omitted to simplify the notation. Therefore, formally, a partition P can be defined[1] as a tuple $P = \{\tau, R\}$ where:

- $\tau = \{\tau_0, \tau_1, .., \tau_n - 1\}$ is a set of n tasks. A task τ_i is characterized by a tuple $\tau_i = \{\phi_i, C_i, D_i, T_i\}$ where ϕ_i is the offset, C_i is the worst case computation time, D_i is the relative deadline and T_i is the period. When all parameters in the system are integers, we may assume without loss of generality that all preemptions occur at integer time values. We then assume, for the remainder of this work, that all parameters are indeed integers. Moreover, constrained deadlines are assumed so $D_i \leq T_i$.

- An arbitrary CPU supply R is represented by a sequence of p intervals $I_1, I_2, ..., I_p$. Every I_i / $1 \leq i \leq p$ is a closed interval $I_i = [s_i, e_i]$ repeated every lcm_τ[2], so that $0 \leq s_i < e_i < s_{i+1}$ and $e_p \leq lcm_\tau$.

  Therefore, $\forall t$ exists a unique interval I_i so that $s_i \leq t \leq e_i$. The CPU supply R for a partition determines the *p* temporal slots in which tasks allocated to the partition are allowed to execute.

The problem to solve is concerns the schedulability of the partition, that is, if task set τ can be scheduled without deadline misses in the slots defined by R.

[1] In the definition of the partition we omit all non-temporal resources

[2] Least Common Multiple of $T_1, .., T_n$

3.2.1 Supply bound function

Although we have characterized R as a set of intervals, it can also be represented graphically.

Figure 3.3 shows two possible CPU supplies for the example of Figure 3.2 with a periodic supply R=(θ, π), where the global level provides θ units of time each π units. In the figure $\theta = 3$ and $\pi = 10$ so both supplies are non-decreasing functions that grow with a slope of 45 degrees at least 3 units every 10 units. $wcsbf_R(t)$ represents the worst case behaviour because provides the 3 units of CPU as late as possible while $sbf_R(t)$ represents other specific allocation of the periodic supply. Therefore, we call the function that represents any specific allocation, the supply bound function of R ($sbf_R(t)$). In this case, note that $sbf_R(t)$ totally coincide with the slots allocation of Figure 3.2 in the sense that the partition is allowed to execute only when $sbf_R(t)$ function increases.

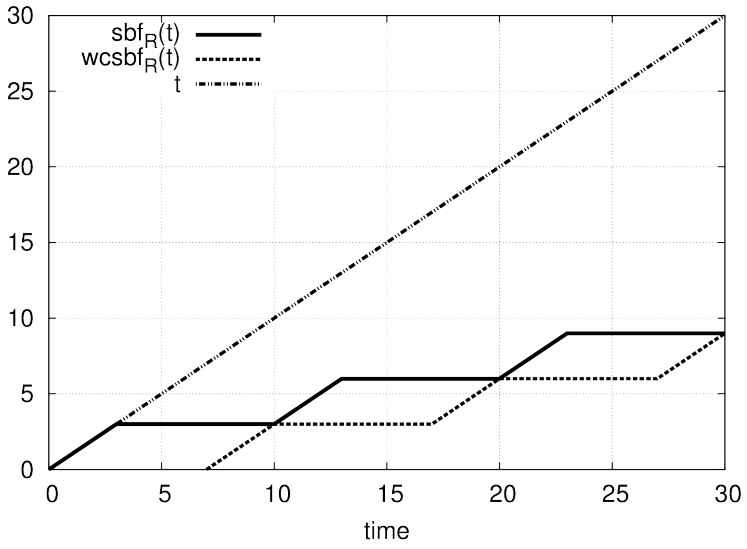


Figure 3.3: Periodic supply bound functions ($\theta = 3, \pi = 10$)

Given a CPU supply R and an interval of length *t*, the supply bound function gives the amount of resource that model R is guaranteed to supply in any time interval of length *t* [**?**]. We can define the supply bound function of R, accordingly with the above definition.

Definition 3.2.1. The supply bound function ($sbf_R(t)$) of an arbitrary supply R expressed as a set of intervals is:

$$sbf_R(t) = \begin{cases} \sum_{i=0}^{j}(e_i - s_i) + t - s_j & \text{if } \exists j / t \in [s_j, e_j], \\ \\ \sum_{i=0}^{j}(e_i - s_i) & \text{if } \exists j / e_j < t < s_{j+1}. \end{cases}$$

, where s_i is the starting point of an interval and e_i is its ending point. From s_i to e_i, the tasks of the partition can be executed. In Figure 3.3, these intervals correspond with the intervals where $sbf_R(t)$ increases. Then, a CPU supply R can be characterized either by a set of intervals I_i or by its $sbf_R(t)$.

Moreover, the following definitions will be used in the next sections. All these properties have been extensively defined in Section 2 and here a quick review is being stated.

The function $G_\tau(t)$ stated in Definition 2.2.1 represents the computation time demanded from initial time to time *t* for a tasks set τ. It can be calculated as:

$$G_\tau(t) = \sum_{i=1}^{n} C_i \cdot \left\lceil \frac{t}{T_i} \right\rceil$$

If tasks are simultaneously activated at time t = 0 (i.e. $\phi_i = 0$ for all the tasks so the task set is synchronous), then:

In Definition 2.2.2 [**?**] [**?**], the maximum cumulative execution time requested by jobs of τ whose absolute deadlines are less than or equal to *t* is called *demand bound function* and is stated as:

$$dbf_\tau(t) = \sum_{i=1}^{n} C_i \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor$$

This function only increases in the so-called *scheduling points* that is, when a deadline arrives.

To generalize, when the task set is asynchronous (i.e. $\exists \phi_i \neq 0$), the processor demand function in interval $[t_1, t_2)$ is defined as:

Definition 3.2.2. [**?**] [**?**]

$$dbf(t_1, t_2) = \sum_{i=1}^{n} \eta_i(t_1, t_2) C_i$$

where:

$$\eta_i(t_1, t_2) = max\{0, (\left\lfloor \frac{t_2 - \phi_i - D_i}{T_i} \right\rfloor - \left\lceil \frac{t_1 - \phi_i}{T_i} \right\rceil + 1)\}$$

From now on, let us assume the task set is synchronous. Therefore, definition 2.2.2 will be used to deduct the minimum supply bound function, in spite of the possibility of using definition 3.2.2 to obtain any other demand function.

3.3 Schedulable CPU supply functions

In this section, specific supply bound functions $sbf_R(t)$ that ensure the schedulability of task sets, τ, are being defined. Specifically, two functions are obtained: $gsbf_\tau(t)$ and $msbf_\tau(t)$.

3.3.1 Schedulable sbf$_\tau$(t) based on G(t)

This section presents the demanded computation supply function, $gsbf_\tau(t)$. This function gives a schedulable supply for τ. We will base our method on the $G_\tau(t)$ function (Definition 2.2.1).

Definition 3.3.1. A characteristic point, t_j, of $G_\tau(t)$ is the one complying with:

$$G_\tau(t_j - \epsilon) < G_\tau(t_j + \epsilon) \quad 0 \leq t_j \leq lcm_\tau \quad \forall \epsilon \to 0$$

, so that t_j coincides with the activation of $\tau_i \in \tau$.

Property 3.3.1. [**?**] Let τ be a schedulable task set. Let t_x be an instant t_x such that:

$$G_\tau(t_x) \leq t_x$$

Then, the processor must have been idle for at least $t_x - G_\tau(t_x)$ time units from initial time.

From Definition 2.2.1 and Property 3.3.1, the demanded computation supply function, $gsbf_\tau(t)$, is presented.

Definition 3.3.2. The $gsbf_\tau(t)$ is defined as:

$$gsbf_\tau(t) = \begin{cases} t - \sum_{i=0}^{j-1}(s_{i+1} - e_i) & \text{if } \exists j / t \in [s_j, e_j], \\ \\ G_\tau(t) & \text{if } \exists j / e_j < t < s_{j+1}. \end{cases}$$

, where:

$$\begin{aligned} s_j &= t_j + \theta_j \\ e_j &= t_j - G_\tau(t_j - \epsilon) + G_\tau(t_j + \epsilon) + \theta_j \\ \theta_j &= max\{0, (e_{j-1} - t_j)\} \end{aligned}$$

and each t_j is an characteristic point of $G_\tau(t_j)$.

Figure 3.4 shows how the function is obtained graphically in the first intervals. In this Figure, the characteristic points are depicted ($t_1, t_2, ...$) and, applying Definition 3.3.2, the start and end points of the intervals of $gsbf_\tau(t)$ are calculated. They correspond to the intervals where $gsbf_\tau(t)$ increases.

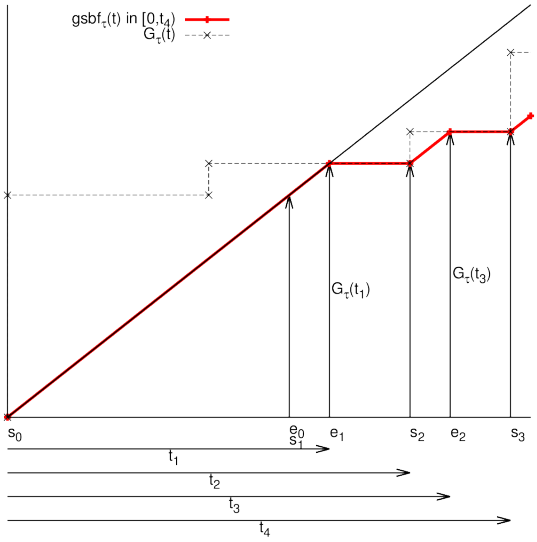


Figure 3.4: Calculation of $gsbf_\tau(t)$ in $[0, t_4)$

To obtain a more compact definition, let us replace the values of s_j and e_j:

$$\begin{aligned}\sum_{i=0}^{j-1}(s_{i+1} - e_i) &= (s_1 - e_0) + (s_2 - e_1) + ... + (s_j - e_{j-1}) \\ &= t_1 + \theta_1 - (t_0 - G_\tau(t_0 - \epsilon) + G_\tau(t_0 + \epsilon) + \theta_0 + \\ &+ t_2 + \theta_2 - (t_1 - G_\tau(t_1 - \epsilon) + G_\tau(t_1 + \epsilon) + \theta_1 + \\ &+ ... + \\ &+ t_{j-1} + \theta_{j-1} - (t_{j-1} - G_\tau(t_{j-2} - \epsilon) + \\ &+ G_\tau(t_{j-2} + \epsilon) + \theta_{j-2}) + \\ &+ t_j + \theta_j - (t_j - G_\tau(t_{j-1} - \epsilon) + G_\tau(t_{j-1} + \epsilon) + \theta_{j-1}) + \\ &= -t_0 + G_\tau(t_0 - \epsilon) - \theta_0 + t_j + \theta_j - G_\tau(t_{j-1} + \epsilon)\end{aligned}$$

As $t_0 = G_\tau(t_0 - \epsilon) = \theta_0 = 0$ and $G_\tau(t_{j-1} + \epsilon) = G_\tau(t_j - \epsilon)$, then, one more

compact definition of $gsbf_\tau(t)$ is:

$$gsbf_\tau(t) = \begin{cases} t - t_j - \theta_j + G_\tau(t_j - \epsilon) & \text{if } \exists j / t \in [s_j, e_j], \\ \\ G_\tau(t) & \text{if } \exists j / e_j < t < s_{j+1}. \end{cases}$$

Once $gsbf_\tau(t)$ has been defined, the schedulability of τ under this CPU supply can be demonstrated. For this reason, the concept of initial critical interval (ICI) must be remembered. ICI [**?**] (also called synchronous busy period in Theorem 2.2.3) is defined as the temporal interval between initial time and the first instant R, when all requests have already been served and no additional requests have arrived yet, assuming that the processor is not idle while tasks are pending. Therefore, this range is [0, R).

Lemma 3.3.1. [**?**] [**?**] Let τ be a synchronous periodic task set and [0,R) its initial critical interval (ICI). τ is schedulable if and only if it can be scheduled in ICI.

Next theorems are used to construct the $gsbf_\tau(t)$ interval by interval and, in each interval, the condition $gsbf_\tau(t) \leq t$ will be checked.

Theorem 3.3.1. *Let $t_1 \in \mathbb{N}$ so that:*

$$\min_t t_1 : G_\tau(t_1) \leq t_1 \quad \forall t \in (0, lcm_\tau]$$

Then,

$$gsbf(t) = t \quad \textit{if } t \in [0, t_1)$$

Proof. Because of the definition of ICI, in the range [0,R) there is no CPU idle time. For this reason, from 0 to t_1, the processor must be always busy. So, $t_1 = R$. □

Theorem 3.3.2. *Let $t_2 \in \mathbb{N}$ so that:*

$$\min_t t_2 : G_\tau(t_2 - \epsilon) \leq G_\tau(t_2 + \epsilon) \quad t_1 < t_2 \leq lcm_\tau \quad \forall \epsilon \to 0$$

, that is, t_2 is a characteristic point of $G_\tau(t)$.

Then,

$$gsbf_\tau(t) = \begin{cases} t & \textit{if } t \in [0, t_1), \\ G_\tau(t) & \textit{if } t \in [t_1, t_2). \end{cases}$$

Proof. The proof is based on the addition of a new task τ_{n+1}, which will be executed only when CPU is idle, that is, when τ is not executing. This time corresponds to the interval $[t_1, t_2)$. Let

$$\tau' = \tau \bigcup \tau_{n+1}$$

where

$$\begin{aligned}
\phi_{n+1} &= t_1 \\
C_{n+1} &= t_2 - t_1 \\
D_{n+1} &= t_2 \\
T_{n+1} &= max\{\phi_1, ..., \phi_N, \phi_{n+1}\} + 2 \cdot lcm_\tau \text{[?]} \\
&= \phi_{n+1} + 2 \cdot lcm_\tau
\end{aligned}$$

Once this task has been added, let's check that the first interval, R, when all requests have already been served and no additional requests have arrived yet, is $[0, t_2)$.

$$\begin{aligned}
G_{\tau'}(t) &= G_\tau(t) + G_{\tau^{n+1}}(t) \\
&= \sum_{i=1}^{n} C_i \left\lceil \frac{t}{T_i} \right\rceil + (t_2 - t_1) \left\lceil \frac{t}{\phi_{n+1} + 2 \cdot lcm_\tau} \right\rceil \\
&= \sum_{i=1}^{n} C_i \left\lceil \frac{t}{T_i} \right\rceil + (t_2 - t_1)
\end{aligned}$$

When $t = t_2 - \epsilon$, $G_\tau(t) = t_1$ the result of the previous equation is:

$$\begin{aligned}
G_{\tau'}(t_2 - \epsilon) &= G_\tau(t_2 - \epsilon) + G_{\tau^{n+1}}(t_2 - \epsilon) \\
&= \sum_{i=1}^{n} C_i \left\lceil \frac{t_2 - \epsilon}{T_i} \right\rceil + (t_2 - t_1) \\
&= t_1 + (t_2 - t_1) \\
&= t_2
\end{aligned}$$

It is clear that, adding this new task set, $G_{\tau'}(t) = t$ and the new ICI is $[0, t_2)$. So, as a result of Lemma 3.3.1, to prove schedulability of τ', the condition $dbf_{\tau'}(t) \leq t$ must be held in all the scheduling points in $[0, t_2)$.

Let us assume that t_x is a scheduling point in $[0, t_2)$. As $t_x < D_{n+1}$ clearly $dbf_{\tau'}(t_x) = dbf_{\tau}(t_x) \leq t_x$. Therefore, the schedulability of τ has been demonstrated because of its schedulability in ICI.

□

As a result of the previous theorem, we derive the next interval.

Lemma 3.3.2. Let $t_3, t_4 \in \mathbb{N}$ so that:

$$t_3 : G_\tau(t_2 + \epsilon) - G_\tau(t_2 - \epsilon) + t_2$$
$$t_4 : G_\tau(t_4 - \epsilon) \leq G_\tau(t_4 + \epsilon)$$

, where $t_2 \leq t_3 \leq t_4 \leq lcm_\tau, \quad \forall \epsilon \to 0$

Then,

$$gsbf_\tau(t) = \begin{cases} t & \text{if } t \in [0, t_1), \\ G_\tau(t) & \text{if } t \in [t_1, t_2), \\ t - (t_2 - t_1) & \text{if } t \in [t_2, t_3), \\ G_\tau(t) & \text{if } t \in [t_3, t_4). \end{cases}$$

Proof. Following the same reasoning as Theorem 3.3.2, we add a new task whose computation time coincides:

$$\tau'' = \tau \bigcup \tau_{n+1} \bigcup \tau_{n+2}$$

where

$$\begin{aligned} \phi_{n+1} &= t_1 \\ C_{n+1} &= t_2 - t_1 \\ D_{n+1} &= t_2 \\ T_{n+1} &= max\{\phi_1, ..., \phi_N, \phi_{n+1}, \phi_{n+2}\} + 2 \cdot lcm_\tau \\ &= \phi_{n+2} + 2 \cdot lcm_\tau \end{aligned}$$

$$\begin{aligned} \phi_{n+2} &= t_3 \\ C_{n+2} &= t_4 - t_3 \\ D_{n+2} &= t_4 \\ T_{n+2} &= max\{\phi_1, ..., \phi_N, \phi_{n+1}, \phi_{n+2}\} + 2 \cdot lcm_\tau \\ &= \phi_{n+2} + 2 \cdot lcm_\tau \end{aligned}$$

Once this task has been added, let us calculate the ICI in this new scenario:

$$\begin{aligned} G_{\tau''}(t) &= G_{\tau}(t) + G_{\tau^{n+1}}(t) + G_{\tau^{n+2}}(t) \\ &= \sum_{i=1}^{n} C_i \left\lceil \frac{t}{T_i} \right\rceil + (t_2 - t_1) \left\lceil \frac{t}{\phi_{n+2} + 2 \cdot lcm_{\tau}} \right\rceil + \\ &+ (t_4 - t_3) \left\lceil \frac{t}{\phi_{n+2} + 2 \cdot lcm_{\tau}} \right\rceil \\ &= \sum_{i=1}^{n} C_i \left\lceil \frac{t}{T_i} \right\rceil + (t_2 - t_1) + (t_4 - t_3) \end{aligned}$$

When $t = t_4 - \epsilon$, $G_{\tau}(t) = t_1 + (t_3 - t_2)$. Therefore:

$$\begin{aligned} G_{\tau''}(t_4 - \epsilon) &= G_{\tau}(t_4 - \epsilon) + G_{\tau^{n+1}}(t_4 - \epsilon) + G_{\tau^{n+2}}(t_4 - \epsilon) \\ &= \sum_{i=1}^{n} C_i \left\lceil \frac{t_4 - \epsilon}{T_i} \right\rceil + (t_2 - t_1) + (t_4 - t_3) \\ &= t_1 + (t_3 - t_2) + (t_2 - t_1) + (t_4 - t_3) \\ &= t_4 \end{aligned}$$

It can be concluded that, adding a new task set,τ_{n+2}, $G_{\tau''}(t) = t$ in $t = t_4$ so the new ICI is $[0, t_4)$. So, as a result of Lemma 3.3.1, to prove schedulability of τ'', the condition $dbf_{\tau''}(t) \leq t$ must be held in all the scheduling points in $[0, t_4)$.

Let us assume that t_x is a scheduling point in $[0, t_4)$. As $t_x < D_{n+2}$, in Theorem 3.3.2 it has been demonstrated that $dbf_{\tau''}(t_x) = dbf_{\tau'}(t_x) \leq t_x$. So, τ'' is schedulable in the hyperperiod because of its schedulability in ICI.

□

From these first intervals, the complete definition of the demanded computation supply function can be built recursively. Values of s_j and e_j are deduced according to the different shapes of the function, depending on how $G_{\tau}(t)$ is built.

The algorithm that implements the slot construction is presented in Listing 3.1.

The $gsbf_{\tau}(t)$ represents a set of temporal windows that can successfully schedule τ. Many sets of slots fulfil this purpose but $gsbf_{\tau}(t)$supplies the time only when a task is activated, that is, as soon as possible. And for this reason, the resulting schedule exactly coincides with the schedule resulting from assigning all CPU time to the partition.

Listing 3.1: $gsbf_\tau(t)$ algorithm

```
1  function gsbf(τ) is
2      j,e_j,s_j,t_0,t_1=0;
3      ε → 0;
4      while(t_1<lcm_τ) loop
5          if G(t_0 − ε) < G(t_0 + ε) then
6              θ_0 = max {0, e_j − t_0};
7              s_j = t_0 + θ_0;
8              e_j = t_0 − G(t_0 − ε) + G(t_0 + ε) + θ_0;
9          end if ;
10         t_0++; j++;
11     end while;
12 end gsbf;
```

3.3.1.1 Example of gsbf$_\tau$(t) use

Let's consider a partition with three tasks ($\tau = \{\tau_0, \tau_1, \tau_2\}$). Task parameters are listed in Table 3.1.

Table 3.1: Task parameters τ

	C_i	D_i	T_i
τ_0	1	4	5
τ_1	6	10	15
τ_2	5	21	30

The definition of $gsbf_\tau(t)$ in definition 3.3.2 is used to calculate the function. The first step consists in calculating $G_\tau(t)$ and all its characteristic points in $[0, lcm_\tau]$. Table 3.2 shows all these values.

Table 3.2: Characteristic points of $gsbf_\tau(t)$

t	0	5	10	15	20	25
$G_\tau(t)$	12	13	14	21	22	23

Now, applying Definition 3.3.2, s_i and e_i are calculated within $[0, lcm_\tau]$ and represented in Table 3.3. The last step is to calculate $gsbf_\tau(t)$ inside each inter-

val. Once it has been calculated, the representation of the function is shown in Figure 3.5.

Table 3.3: Definition of $[s_i, e_i]$

i	0	1	2	3	4	5
s_i	0	12	13	15	22	25
e_i	12	13	14	22	23	26

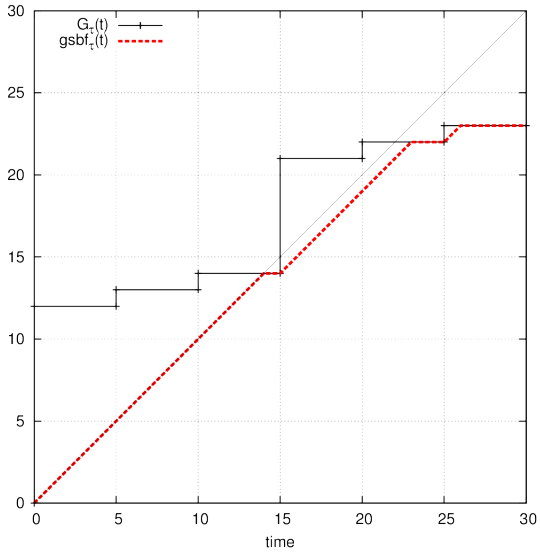


Figure 3.5: Representation of $gsbf_\tau(t)$

As seen, CPU will be busy in the intervals where $gsbf_\tau(t)$ grows. These intervals are defined in Table 3.4.

The execution chronogram of the task set considering that the CPU supply R coincides with the demanded computation supply function is depicted in Figure 3.6. As seen, the task set is schedulable in this slot assignment.

Thus, the schedulability of the $gsbf_\tau(t)$ function for a task set and the methodology for calculating it have been demonstrated.

3.3.2 Schedulable $sbf_\tau(t)$ based on $dbf_\tau(t)$

In Section 3.3.1, a valid supply that gives CPU when tasks are activated has been presented. Now, another valid supply is being presented but, in this new situation,

Table 3.4: Definition of $[s_i, e_i]$

	s_i	e_i
I_1	0	14
I_2	15	23
I_3	25	26

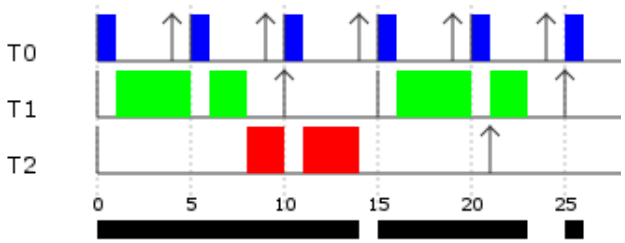


Figure 3.6: Execution chronogram of τ with $gsbf_\tau(t)$ in Table 3.4

CPU is supplied just before the deadlines arrive. If $gsbf_\tau(t)$ consists on supplying CPU as soon as possible (for task activation), $msbf_\tau(t)$ will consist in supplying CPU as late as possible.

To obtain $msbf_\tau(t)$, our method is based on the demand bound function for a task set. $msbf_\tau(t)$ is built by intervals and is proved that in each interval there are no deadline misses. Thus, $msbf_\tau(t)$ will be generalized.

Theorem 3.3.3. *Let $t_1 \in \mathbb{N}$ so that:*

$$t_1 - dbf_\tau(t_1) = \min_t(t - dbf_\tau(t)) \quad \forall t \in (0, lcm_\tau]$$

And,

$$msbf_\tau(t) = \begin{cases} t - t_1 + dbf_\tau(t_1) & \text{if } t \in [t_1 - dbf_\tau(t_1), t_1], \\ 0 & \text{if } t \in [0, t_1 - dbf_\tau(t_1)). \end{cases}$$

If $sbf_R(t) = msbf_\tau(t)$ then τ is schedulable.

Proof. The proof is based on adding a new task τ_{n+1}. This task can only be executed when τ is not allowed to execute, that is, in $[0, t_1 - dbf_\tau(t_1))$ and is demonstrated that the new set is schedulable. Then, computation time of τ_{n+1} is increased in order to check that the new set is not schedulable.

Let

$$\tau' = \tau \bigcup \tau_{n+1}$$

where

$$\begin{aligned} C_{n+1} &= t_1 - dbf_\tau(t_1) \\ D_{n+1} &= t_1 - dbf_\tau(t_1) \\ T_{n+1} &= max\{\phi_1, ..., \phi_N\} + 2 \cdot lcm_\tau \\ \phi_{n+1} &= 0 \end{aligned}$$

To prove schedulability of τ', the condition $dbf_{\tau'}(t) \leq t$ must be met in all the scheduling points in $[0, t_1]$. Let us assume that a is a scheduling point in $[0, t_1]$. If $a < D_{n+1}$, obviously $dbf_{\tau'}(a) = dbf_\tau(a) \leq a$. If $a \geq D_{n+1}$ the demand bound function of the new task set τ' is:

$$\begin{aligned} dbf_{\tau'}(a) &= dbf_\tau(a) + C_{n+1} \\ &= dbf_\tau(a) + t_1 - dbf_\tau(t_1) \end{aligned}$$

As $t_1 - dbf_\tau(t_1) = \min_t(t - dbf_\tau(t))$ then

$$t_1 - dbf_\tau(t_1) \leq (a - dbf_\tau(a))$$

So,

$$\begin{aligned} dbf_{\tau'}(a) &\leq dbf_\tau(a) + a - dbf_\tau(a) \\ &\leq a \end{aligned}$$

Now, let us assume

$$\tau'' = \tau \bigcup \tau_{n+1}$$

and

$$\begin{aligned} C_{n+1} &= t_1 - dbf_\tau(t_1) + \epsilon \\ D_{n+1} &= t_1 - dbf_\tau(t_1) + \epsilon \\ T_{n+1} &= max\{\phi_1, ..., \phi_N\} + 2 \cdot lcm_\tau \\ \phi_{n+1} &= 0 \end{aligned}$$

being ϵ a small positive number such that $0 < \epsilon \leq 1$.

Following the same reasoning:

$$\begin{aligned} dbf_{\tau'}(t_1) &= dbf_\tau(t_1) + t_1 - dbf_\tau(t_1) + \epsilon \\ &= t_1 + \epsilon \end{aligned}$$

so τ'' is not schedulable.

□

As a result of the previous theorem, $msbf_\tau(t)$ until t_1, expressed as a set of intervals, is $msbf_\tau(t) = I_0 = [t_1 - dbf_\tau(t_1), t_1]$. Using a similar approach the next interval will be derived.

Lemma 3.3.3. Let $t_2 \in \mathbb{N}$, $t_1 < t_2$ so that:

$$t_2 - dbf_\tau(t_2) = \min_t(t - dbf_\tau(t)) \quad \forall t \in (t_1, lcm_\tau]$$

And

$$msbf_\tau(t) = \begin{cases} t - t_1 + dbf_\tau(t_1) & \text{if } t \in [t_1 - dbf_\tau(t_1), t_1], \\ dbf_\tau(t_1) & \text{if } t \in (t_1, \\ & t_2 - dbf_\tau(t_2) + dbf_\tau(t_1)) \\ t - t_2 + dbf_\tau(t_2) & \text{if } t \in [t_2 - dbf_\tau(t_2) \\ & +dbf_\tau(t_1), t_2], \\ 0 & \text{if } t \in [0, t_1). \end{cases}$$

If $sbf_R(t) = msbf_\tau(t)$ then τ is schedulable.

Proof. Schedulability in $[0, t_1]$ is assured due to Theorem 3.3.3. Following the same reasoning as Theorem 3.3.3, a task whose computation time coincides with the idle time between I_0 and SI_1 is added and its deadline is equal to the start time of I_1 (s_1).

Let

$$\tau' = \tau \bigcup \tau_{n+1} \bigcup \tau_{n+2}$$

where

$$\begin{aligned} C_{n+1} &= t_1 - dbf_\tau(t_1) \\ D_{n+1} &= t_1 - dbf_\tau(t_1) \\ T_{n+1} &= max\{\phi_1, ..., \phi_N\} + 2 \cdot lcm_\tau \\ \phi_{n+1} &= 0 \end{aligned}$$

$$
\begin{aligned}
C_{n+2} &= t_2 - dbf_\tau(t_2) + dbf_\tau(t_1) - t_1 \\
D_{n+2} &= t_2 - dbf_\tau(t_2) + dbf_\tau(t_1) \\
T_{n+2} &= max\{\phi_1, ..., \phi_N\} + 2 \cdot lcm_\tau \\
\phi_{n+2} &= t_1
\end{aligned}
$$

Let us assume that a is a scheduling point in $(t_1, t_2]$. If $a < D_{n+2}$, then $dbf_\tau(a) = dbf_{\tau'}(a)$, so the new task set is schedulable. If $a \geq D_{n+2}$, following the same reasoning as in Theorem 3.3.3, the computation time of τ_{n+1} and τ_{n+2} is added to $sbf_{\tau'}(t)$:

$$
\begin{aligned}
dbf_{\tau'}(a) &= dbf_\tau(a) + t_1 - dbf_\tau(t_1) + \\
&\quad + t_2 - dbf_\tau(t_2) + dbf_\tau(t_1) - t_1 \\
&= dbf_\tau(a) + t_2 - dbf_\tau(t_2)
\end{aligned}
$$

Given

$$t_2 - dbf_\tau(t_2) \leq (a - dbf_\tau(a))$$

that

$$
\begin{aligned}
dbf_{\tau'}(a) &\leq dbf_\tau(a) + a - dbf_\tau(a) \\
&\leq a
\end{aligned}
$$

□

Theorem 3.3.3 and Lemma 3.3.3 provide a method for obtaining the first two intervals of $msbf_\tau(t)$ function and it has been proved that this function is schedulable. Figure 3.7 shows graphically how the function is obtained.

It is straightforward to recursively construct all the minimum supply slots needed by τ to maintain feasibility: finding t_j points in where it holds that $t_j - dbf_\tau(t_j)$ is the minimum value in $(t_{j-1}, lcm_\tau]$. These points are called the *minimum scheduling points* t_j. Therefore, the $msbf_\tau(t)$ is defined as in Definition 3.2.1 but now specific values are given to s_j and e_j:

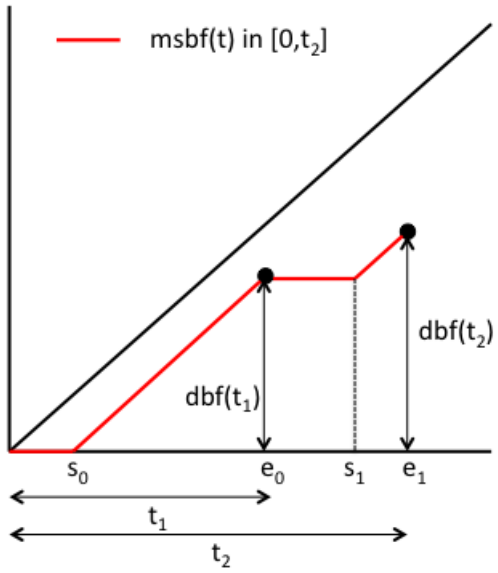


Figure 3.7: Calculation of $msbf_\tau(t)$ in $[0, t_2]$

$$msbf_\tau(t) = \begin{cases} \sum_{i=0}^{j}(e_i - s_i) + t - e_j & \text{if } \exists j / t \in [s_j, e_j], \\ \\ \sum_{i=0}^{j}(e_i - s_i) & \text{if } \exists j / e_j < t < s_{j+1}. \end{cases}$$

where $s_j = t_j - dbf_\tau(t_j) + dbf(t_{j-1})$ and $e_j = t_j$.
Replacing the values of s_j and e_j in the previous definition:

$$\sum_{i=0}^{j}(e_i - s_i) = t_1 - t_1 + dbf_\tau(t_1) - dbf(t_0) +$$
$$+t_2 - t_2 + dbf_\tau(t_2) - dbf(t_1) + ...$$

Assuming that $t_0 = 0$ and $dbf_\tau(0) = 0$:

$$\sum_{i=0}^{j}(e_i - s_i) = dbf_\tau(t_j)$$

Therefore, a more compact definition for $msbf_\tau(t)$ is provided.

Definition 3.3.3. The definition of the minimum supply bound function $msbf_\tau(t)$ is:

$$msbf_\tau(t) = \begin{cases} t - t_j + dbf_\tau(t_j) & \text{if } \exists j / t \in [s_j, e_j], \\ \\ dbf_\tau(t_j) & \text{if } \exists j / e_j < t < s_{j+1}. \end{cases}$$

The algorithm that implements the slot construction is presented in Listing 3.2.

Listing 3.2: $msbf_\tau(t)$ algorithm

```
1  function msbf(τ) is
2      i , e_i,s_i,t_1,t_2=0;
3      while(t_2<lcm_τ) loop
4          t_2= min_{e_i<t≤lcm_τ} (dbf_τ(t));
5          s_i = t_2 − dbf_τ(t_2) + dbf_τ(t_1);
6          e_i = t_2;
7          t_1 = t_2;
8          i++;
9      end while;
10 end msbf;
```

The previous function is obtained from $dbf_\tau(t)$ in Definition 2.2.2, particularized for synchronized tasks. If we assume the possibility of asynchronism between tasks (i.e., $\phi_i \neq 0$), this algorithm is also valid due to the inclusion of the offset in Definition 3.2.2. If a task set is synchronous, Definition 2.2.2 will be applied to obtain $dbf_\tau(t)$ and, consequently, $msbf_\tau(t)$. However, if any task does not start at the same time as others, then another $dbf_\tau(t)$ will be obtained because of this offset and, consequently, other $msbf_\tau(t)$, which will also meet the criteria of schedulability of all tasks.

The previous algorithm works over the entire hyperperiod (lcm_τ), which depending on the values of task periods can be a large value. To overcome this disadvantage, the results presented in [**?**] could be used, where an algorithm to compute the minimum hyperperiod for a set of periodic activities when period is specified as a range is presented . If, in spite of considering periods as specific values, they are treated as ranges of valid values, [**?**] will select the value inside each inter-

val which causes the minimum hyperperiod. This method drastically reduces the hyperperiod.

As noted in the conclusions section, we are working on an upper bound of $msbf(\tau)$ to reduce the complexity of the algorithm.

Once $msbf_\tau(t)$ is obtained, the following theorem provides the schedulability condition of $\{\tau, R\}$.

Theorem 3.3.4. *A task set τ is schedulable under a CPU supply R if and only if:*

$$\forall t \quad sbf_R(t) = msbf_\tau(t)$$

Proof. It is proved for any time point a :

$$msbf_\tau(a) \geq dbf_\tau(a) \quad \forall a \in [0, lcm_\tau]$$

Two cases are assumed:

- Case 1: $a \notin [s_j, e_j]$.
- Case 2: $a \in [s_j, e_j]$.

Case 1: If $a \notin [s_j, e_j]$, then $\exists j$ so $e_j < a < s_{j+1}$. Applying the second case in the $msbf_\tau(t)$ definition:

$$msbf_\tau(a) = dbf_\tau(a)$$

Case 2: If $a \in [s_j, e_j]$, applying the first case of the msbf definition:

$$msbf_\tau(a) = a - t_j + dbf_\tau(t_j)$$

As $dbf_\tau(t)$ is a positive and monotonic increasing function it holds that [**?**]:
If $a \leq t_j$ then $dbf_\tau(a) \leq dbf_\tau(t_j)$
And, as τ is schedulable then $dbf_\tau(t_j) - t_j \leq 0$.
Therefore:

$$\begin{aligned} msbf_\tau(a) &\geq dbf_\tau(a) - dbf_\tau(t_j) - t_j \\ &\geq dbf_\tau(a) \end{aligned}$$

In any case: $msbf_\tau(a) \geq dbf_\tau(a)$.

□

3.3.2.1 Example of msbf$_\tau$(t) use

Let us consider a partition with three tasks ($\tau = \{\tau_1, \tau_2, \tau_3\}$). Task parameters are listed in Table 3.1. Figure 3.8 shows the execution chronogram for the task set scheduled under EDF policy if the partition is the only one in the system, that is, the CPU supply is a unique slot $I_0 = [0, 30]$. As the figure shows, the task set is schedulable since there are no missed deadlines throughout the hyperperiod (lcm_τ).

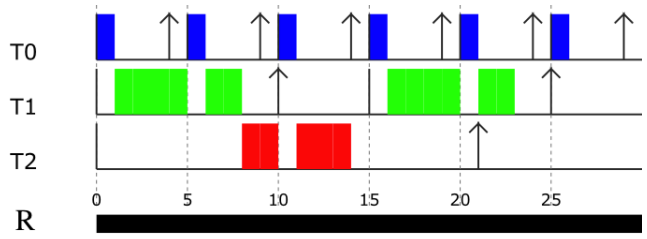


Figure 3.8: Execution chronogram of τ with $R = I_0 = [0, 30]$

Let us consider now that there are more partitions in the system so the SI assigns to the considered partition a CPU supply R which matches the $msbf_\tau(t)$.

To find out whether the CPU supply R is able to successfully schedule the task set, $msbf_\tau(t)$ is obtained. The methodology, as explained in previous sections, consists of calculating the minimum $t - dbf_\tau(t)$ in $[0, lcm_\tau]$. Table 3.5 shows $t - dbf_\tau(t)$ for all the scheduling points in [0,30].

Table 3.5: Scheduling points

t	4	9	10	14	19	21	24	25	29
$dbf_\tau(t)$	1	2	8	9	10	15	16	22	23
$t - dbf_\tau(t$	3	7	2	5	9	6	8	3	6

The scheduling point with the minimum slack ($t - dbf_\tau(t)$) is $t_1 = 10$. Therefore, according to Theorem 3.3.3, the first slot of $msbf_\tau(t)$ is $I_0 = [t_1 - dbf_\tau(t_1), t_1] = [2, 10]$.

In the second iteration, the next scheduling point with the minimum slack must be searched in (10,30]. This point is $t_2 = 25$. Therefore, the next slot of $msbf_\tau(t)$ is $I_1 = [t_2 - dbf_\tau(t_2) + dbf_\tau(t_1), t_2] = [11, 25]$.

In the third and last iteration, it is clear that $t_3 = 29$ as it is the only remaining scheduling point in $(25, 30]$. Therefore $I_2 = [t_3 - dbf_\tau(t_3) + dbf_\tau(t_2), t_3] = [28, 29]$. Table 3.6 summarizes the slots of $msbf_\tau(t)$ and the representation of the function is depicted in Figure 3.9.

Table 3.6: Minimum supply bound

	s_i	e_i
I_1	2	10
I_2	11	25
I_3	28	29

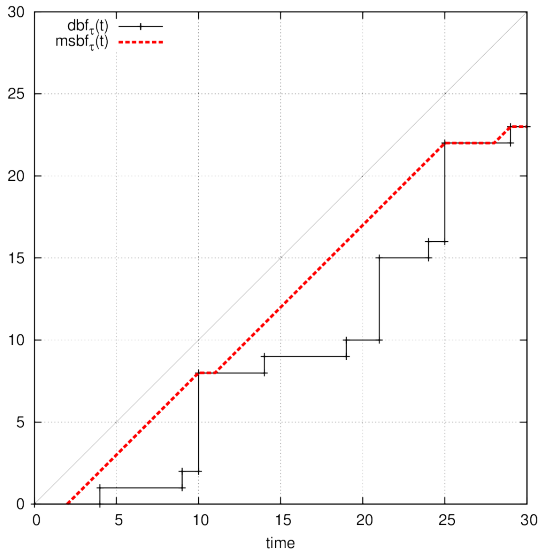


Figure 3.9: Representation of $msbf_\tau(t)$

The execution chronogram of the task set considering that the CPU supply R coincides with the minimum supply calculated is depicted in Figure 3.10. It is seen in the figure that the task set is schedulable and this slots assignment are the more restrictive ones, since if any slot is reduced, task τ_2 will miss its deadline.

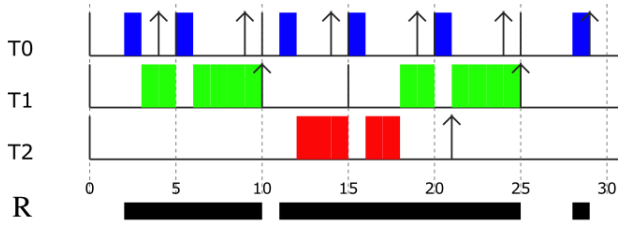


Figure 3.10: Execution chronogram of τ with $R = msbf_\tau(t)$

As $sbf_R(t) = msbf_\tau(t)$, the task set is schedulable. It is depicted in Figure 3.10. Finally, it has become clear the schedulability analysis exposed in section 3.5.

3.4 Schedulable areas

In this section, information about the schedulability of $sbf_R(t)$ depending on its relationship with $gsbf_\tau(t)$ and $msbf_\tau(t)$ is obtained.

Both $gsbf_\tau(t)$ and $msbf_\tau(t)$ are minimum supply functions in the sense that both supply the minimum instants of CPU to assure schedulability of τ.

In Figure 3.11, three zones according to the relation between $msbf_\tau(t)$ and $gsbf_\tau(t)$ are depicted:

- Zone 1: This area depicts CPU supply R functions that are greater than $gsbf_\tau(t)$ and, consequently, greater than $msbf_\tau(t)$.
- Zone 2: This area depicts CPU supply R functions that are between $msbf_\tau(t)$ and $gsbf_\tau(t)$.
- Zone 3: This area depicts CPU supply R functions that are lesser than $msbf_\tau(t)$.

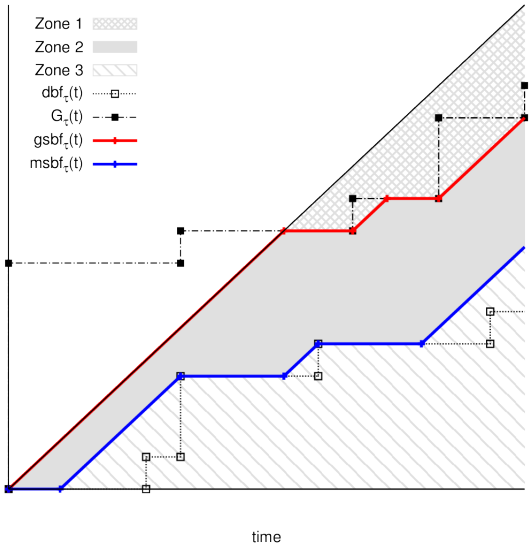


Figure 3.11: Zones according to the position of $msbf_\tau(t)$ and $gsbf_\tau(t)$

For the periodic resource model with D=T (periodic $sbf_R(t)$) any $sbf_R(t)$ located in Zone 1 and 2 is schedulable whereas any $sbf_R(t)$ located in Zone 3 is not ([**?**]). However, as the following counterexamples show, this is no longer true if the slot assignation is arbitrary.

3.4.1 ZONE 1: CPU supply R greater than gsbf$_\tau$(t) and, consequently, msbf$_\tau$(t)

Some might think that any sequence of slots whose characteristic function $sbf_R(t)$ is greater than $gsbf_\tau(t)$ and, consequently, $msbf_\tau(t)$, would assure the schedulability of task set τ. The next example is used to refute this theory.

Let us consider the partition whose tasks are defined in Table 3.1. Let us consider now that there are more partitions in the system so the SI assigns to the considered partition a CPU supply R shown in Table 3.7.

Table 3.7: CPU supply R

	s_i	e_i
I_1	0	5
I_2	7	25
I_3	29	30

The $msbf_\tau(t)$ of the task set is calculated following the description in Section 3.3.2 and, representing in the same graph $msbf_\tau(t)$ and $sbf_R(t)$, it is observed that $sbf_R(t)$ is always above or equal to $msbf_\tau(t)$ (see Figure 3.12).

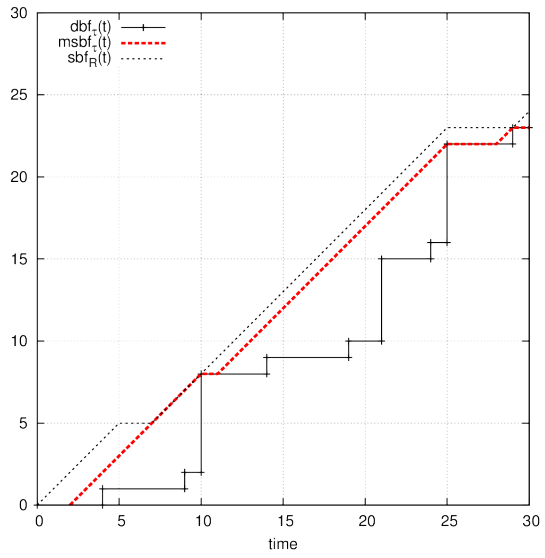


Figure 3.12: Graphical representation of $dbf_\tau(t), msbf_\tau(t)$ and $sbf_R(t)$

However, as can be seen in Figure 3.13, this task set is not schedulable in partition R defined by $sbf_R(t)$. τ_0 misses its deadline in its fifth activation. τ_0 is activated in $time = 25u.t.$ and the next deadline is in $time = 29u.t.$. Inside the interval [25,29], there is no CPU supply R. So, the task set is not schedulable.

Although $sbf_R(t)$ is above $msbf_\tau(t)$, the partition must be defined in such a way as to guarantee the that every task meets the deadlines. If, from the request of a task until the arrival of the corresponding deadline, the task does not have enough time to being executed entirely, the system will not be schedulable, as occurs in the previous situation.

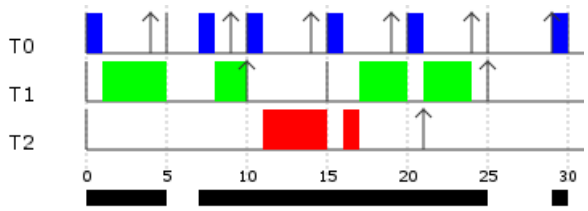


Figure 3.13: Execution chronogram of τ with R in Table 3.7

The same happens with $gsbf_\tau(t)$. Therefore, the idea of ensuring the schedulability of a task set whenever $sbf_R(t)$ is greater than $gsbf_\tau(t)$ is rejected.

Let us consider the task set in Table 3.1 and the CPU supply in Table 3.8. As Figure 3.14 shows, $sbf_{R'}(t)$ is always greater than or equal to $gsbf_\tau(t)$ but, as Figure 3.15 shows, the task set is not schedulable.

Table 3.8: CPU supply R'

	s_i	e_i
I_1	0	25
I_2	29	30

3.4.2 ZONE 2: CPU supply R between msbf$_\tau$(t) and gsbf$_\tau$(t)

Now, let us suppose another scenario: $msbf_\tau(t) \leq sbf_R(t) \leq gsbf_\tau(t)$ used to check that one $sbf_R(t)$, which is defined between $gsbf_\tau(t)$ and $msbf_\tau(t)$, does not necessarily assure the schedulability of the task set. It depends on how the function is defined and if the deadlines are met or not.

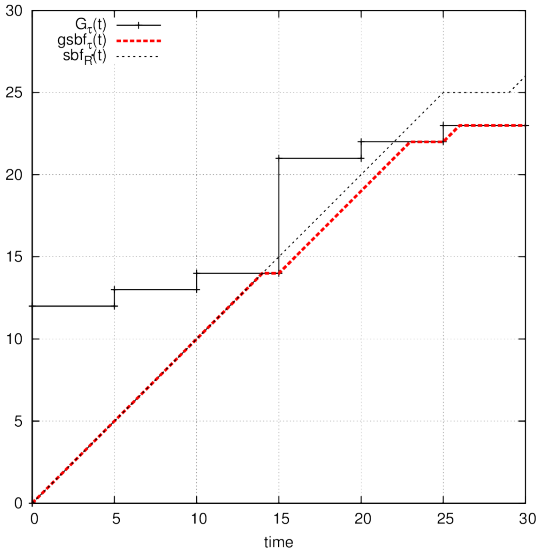


Figure 3.14: Graphical representation of $G_\tau(t)$, $gsbf_\tau(t)$ and $sbf_{R'}(t)$

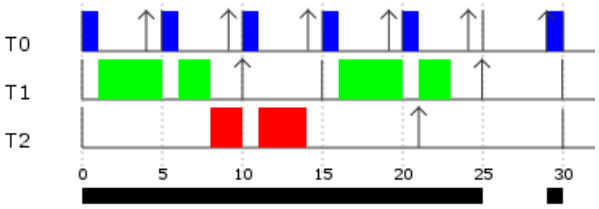


Figure 3.15: Execution chronogram of τ with R' in Table 3.8

Let us define a partition defined by the tasks τ' in Table 3.9 and the CPU supply R" in Table 3.10.

Table 3.9: Task parameters τ'

	C_i	D_i	T_i
τ'_0	2	8	10
τ'_1	5	10	25
τ'_2	7	40	50

Table 3.10: CPU supply R"

	s_i	e_i
I_1	2	16
I_2	21	25
I_3	32	39
I_4	43	44
I_5	45	46

Calculating all the functions as described in previous sections, Figure 3.16 shows $msbf_{\tau'}(t) \leq sbf_{R''}(t) \leq gsbf_{\tau'}(t)\ \forall t$. However, Figure 3.17 shows that τ' is not schedulable, because τ'_1 misses its deadline in its second activation.

3.4.3 ZONE 3: CPU supply R less than msbf$_\tau$(t)

According to the definition of $msbf_\tau(t)$, it is the most restrictive function for scheduling a task set. If any supply CPU R is lesser than $msbf_\tau(t)$, the task set will not be schedulable.

Let us define now a CPU supply, R"', shown in Table 3.11 and task set τ'.

As Figure 3.18 shows, if $sbf_{R'''}(t) \leq msbf_{\tau'}(t)$, in the minimum scheduling points, the CPU supply will not be enough to schedule the tasks. This is because at these points, $sbf_{R'''}(t) \leq dbf_{\tau'}(t)$.

All these counterexamples show that, in contrast to what we might think, the schedulability of a task set is not constrained by being in Zone 2. However, a task set will not be schedulable if its $sbf_R(t)$ is less than $msbf_\tau(t)$, that is, Zone 3.

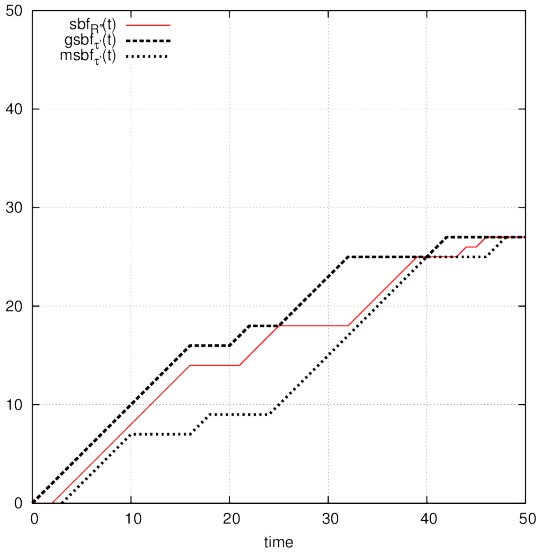


Figure 3.16: Graphical representation of $msbf_{\tau'}(t)$, $gsbf_{\tau'}(t)$ and $sbf_{R''}(t)$

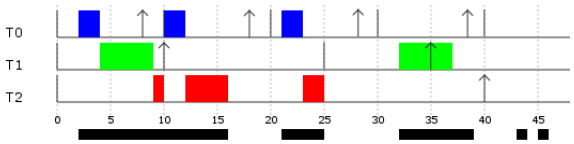


Figure 3.17: Execution chronogram of τ' with R'' in Table 3.10

Table 3.11: CPU supply R"'

	s_i	e_i
I_1	4	10
I_2	12	13
I_3	17	18
I_4	26	30

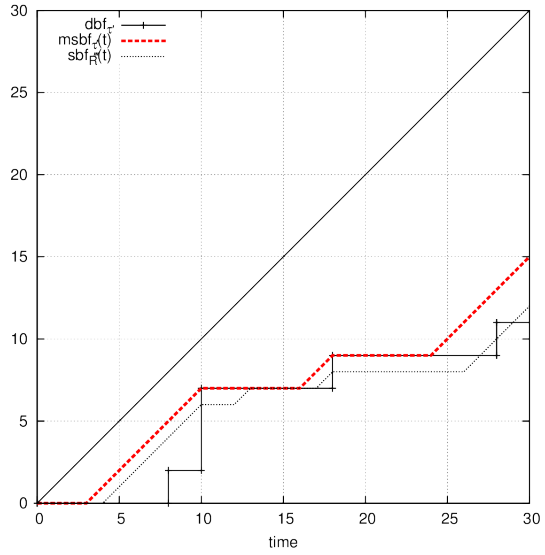


Figure 3.18: Graphical representation of $msbf_{\tau'}(t)$ and $sbf_{R'''}(t)$

As it has been demonstrated, the condition $sbf_R(t) \geq msbf_\tau(t)$ it is not enough because $sbf_R(t)$ can increase at the beginning and have a long idle interval and still be above $msbf_\tau(t)$ and miss deadlines during the long idle interval. However, a periodic resource $R = (\theta, \pi)$ ensures θ units of resource every π units of time. This, jointly with the condition $sbf_R(t) \geq msbf_\tau(t)$ ensures that the processor is assigned to τ whenever it is needed to not miss deadlines.

In an arbitrary CPU supply, it must be ensured that $sbf_R(t)$ provides enough CPU time *from time to time* but not necessarily periodically. With this idea, the objective of the next section is to provide a schedulability test using $gsbf_\tau(t)$ and $msbf_\tau(t)$.

3.5 Schedulability analysis

Theorems 3.5.1 and 3.5.2 provide two alternative schedulability tests.

Let us use the definition of the $sbf_R(t)$ in definition 3.2.1 and $msbf_\tau(t)$ function in definition 3.3.3.

Theorem 3.5.1. *Let $sbf_R(t)$ be a function so that:*

$$msbf_\tau(t) \leq sbf_R(t) \leq t$$

and, moreover, $sbf_R(t)$ increases, at least, at the same time intervals as $msbf_\tau(t)$.

Then, τ is schedulable in R.

Proof. It has been demonstrated that any sequence of slots $sbf_R(t)$ that coincides with $msbf_\tau(t)$ ensures the schedulability of $\{\tau, R\}$ (Section 3.3.2). So, if a sequence of slots supplies CPU time at the same time as $msbf_\tau(t)$ and also supplies it at another extra time, obviously, the task set will also be schedulable.

In other words, any function $sbf_R(t)$ that is above $msbf_\tau(t)$ and, moreover, increases, at least, at the same time intervals as $msbf_\tau(t)$, will be schedulable. Then, the gradient of $sbf_R(t)$ has to be equal to the gradient of $msbf_\tau(t)$ at least in $[s_j, e_j]$ to be schedulable, that is, both functions have to be parallel lines at least in that interval. See Figure 3.19.

The extra time mentioned previously represents CPU idle time.

□

Lemma 3.5.1. If $msbf_\tau(t) \leq sbf_R(t)$ and $sbf_R(t)$ increases, at least, at the same time intervals as $msbf_\tau(t)$, then the number of units of time that the CPU is idle are calculated as follows:

$$CPU\_idle\_time = \sum_{\forall n} [e_n - s_n] - \sum_{\forall j} [e_j - s_j]$$

This situation can be illustrated by the following example. For the task set defined in Table 3.1, the $msbf_\tau(t)$ is obtained in Section 3.3.2.1 and the execution chronogram when $sbf_R(t) = msbf_\tau(t)$ is depicted in Figure 3.10.

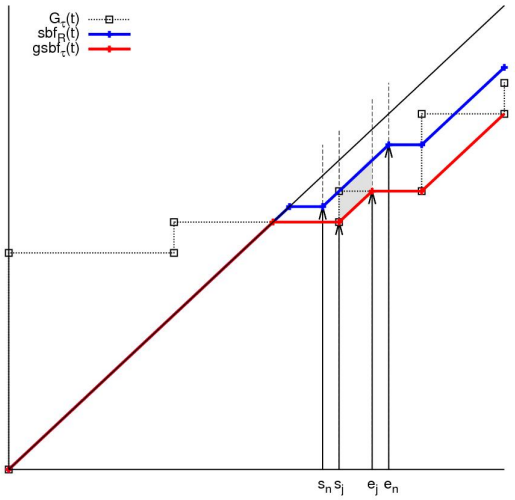


Figure 3.19: Conditions of schedulability in $\{\tau, R\}$

Let us suppose another situation: an $sbf_R(t)$ that is greater than the previously calculated $msbf_\tau(t)$. Moreover, it also increases in the same intervals as $msbf_\tau(t)$ and also in other extra intervals, as shown in Figure 3.20.

In this figure, $msbf_\tau(t)$ is depicted in red and $sbf_R(t)$ is depicted in blue. It is further observed that $sbf_R(t)$ increases at the same time as $msbf_\tau(t)$, that is, time intervals in grey. In addition, there are other time intervals in which $sbf_R(t)$ increases but $msbf_\tau(t)$ does not. They are highlighted in green.

Figure 3.21 shows the task set is schedulable in $\{\tau, R\}$ but there is CPU idle time. In fact, there is as much idle time as time instants when $sbf_R(t)$ increases but $msbf_\tau(t)$ does not.

So, the methodology for determining whether a function $sbf_R(t)$ ensures the schedulability of τ is:

1. Calculate the $msbf_\tau(t)$ for the task set, τ, and express it as set of intervals $I_j = [s_j, e_j]$ where the function increases.

2. Express $sbf_R(t)$ as a set of intervals $I_R = [s_R, e_R]$ in which the function increases.

3. Check if $I_j \subseteq I_R$, that is, $s_R \leq s_j \leq e_j \leq e_R, \forall j, R$.

The same happens with $gsbf_\tau(t)$. Let us use the definition of $gsbf_\tau(t)$ in definition 3.3.2.

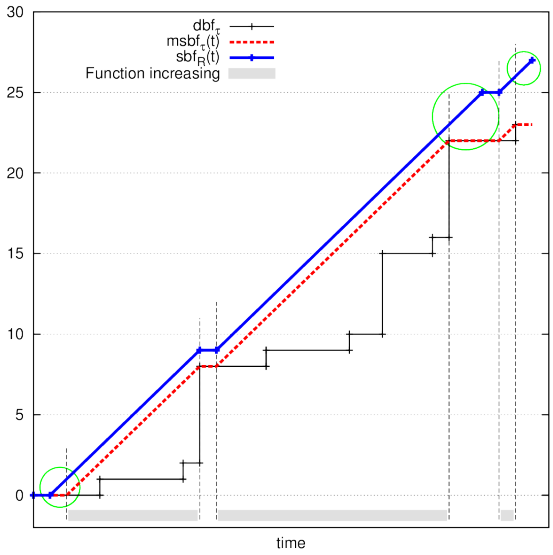


Figure 3.20: Example of schedulability in $\{\tau, R\}$

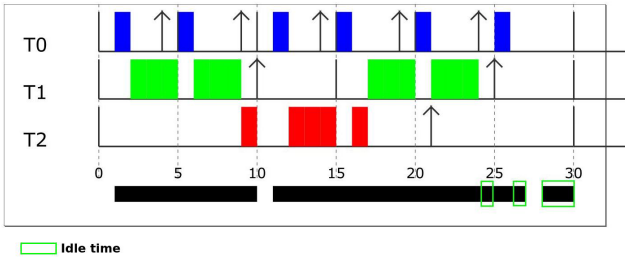


Figure 3.21: Chronogram execution of schedulability in $\{\tau, R\}$

Theorem 3.5.2. *Let $sbf_R(t)$ be a function so that:*

$$gsbf_\tau(t) \leq sbf_R(t) \leq t$$

and, moreover, $sbf_R(t)$ increases, at least, at the same time intervals in as $gsbf_\tau(t)$.

Then, τ is schedulable in R.

Proof. In Section 3.3.1 it has been demonstrated that any sequence of intervals, R, defined by $sbf_R(t)$ that coincides with $gsbf_\tau(t)$ will ensure the schedulability of $\{\tau, R\}$. If, in addition to these intervals, other time instants are added to R, obviously τ will be also schedulable.

In other words: any $sbf_R(t)$ greater than $gsbf_\tau(t)$ and, whose gradient coincides with the gradient of $gsbf_\tau(t)$ in $[s_j, e_j]$ coincides. See Figure 3.22.

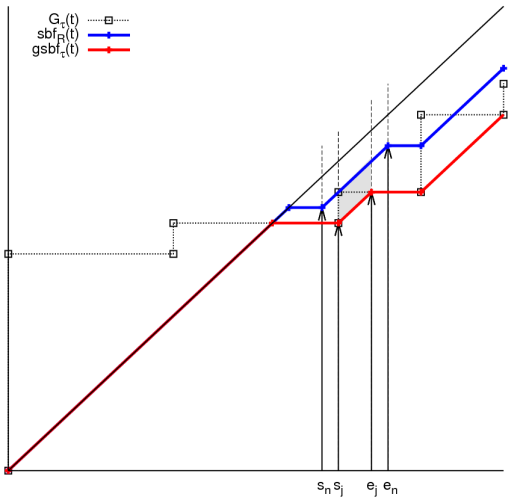


Figure 3.22: Conditions of schedulability in $\{\tau, R\}$

The additional time instants in which $sbf_R(t)$ increases but $gsbf_\tau(t)$ does not, represent the idle time of CPU. □

Lemma 3.5.2. If $gsbf_\tau(t) \leq sbf_R(t)$ and $sbf_R(t)$ increases, at least, at the same time intervals as $gsbf_\tau(t)$, then the number of units of time that the CPU is idle are calculated as follows:

$$CPU\_idle\_time = \sum_{\forall n} [e_n - s_n] - \sum_{\forall j} [e_j - s_j]$$

This situation can be illustrated by the following example. For the task set defined in Table 3.1, the $gsbf_\tau(t)$ is obtained in Section 3.3.1.1 and the execution chronogram when $sbf_R(t) = gsbf_\tau(t)$ is depicted in Figure 3.6.

Let us suppose another situation: an $sbf_R(t)$ that is greater than the previously calculated $gsbf_\tau(t)$. Moreover, it also increases in the same intervals as $gsbf_\tau(t)$ and in other extra intervals, as shown in Figure 3.23.

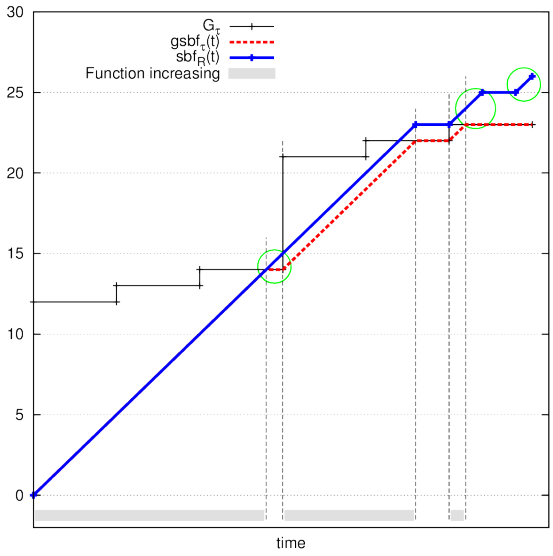


Figure 3.23: Example of schedulability in $\{\tau, R\}$

In this figure, $gsbf_\tau(t)$ is depicted in red and $sbf_R(t)$ is depicted in blue. It is further observed that the $sbf_R(t)$ increases at the same time as $gsbf_\tau(t)$, that is, time intervals in grey. In addition, there are other time intervals in which $sbf_R(t)$ increases but $gsbf_\tau(t)$ does not. They are highlighted in green.

As Figure 3.24 shows, the task set is schedulable in $\{\tau, R\}$ but there is CPU idle time. In fact, there is as much idle time as time instants when $sbf_R(t)$ increases but $gsbf_\tau(t)$ does not.

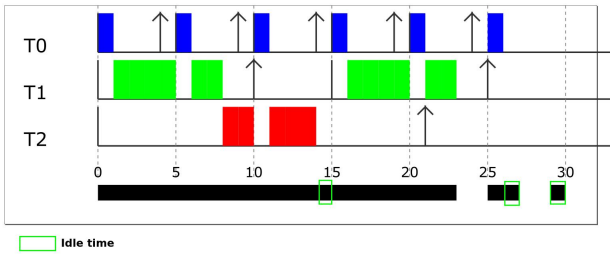


Figure 3.24: Chronogram execution of schedulability in $\{\tau, R\}$

So, the methodology to be followed to determine if a function $sbf_R(t)$ ensures the schedulability of τ is the same as the case of $msbf_\tau(t)$:

1. Calculate the $gsbf_\tau(t)$ for task set, τ, and express it as set of intervals $I_j = [s_j, e_j]$ where the function increases.

2. Express $sbf_R(t)$ as a set of intervals $I_R = [s_R, e_R]$ in which the function increases.

3. Check if $I_j \subseteq I_R$, that is, $s_R \leq s_j \leq e_j \leq e_R, \forall j, R$.

In this section, a condition of schedulability has been proposed. In previous sections, two different schedulability algorithms have been presented ($msbf_\tau(t)$ and $gsbf_\tau(t)$) and, now, a way of checking the schedulability of any sequence of slots by comparing to these functions has been suggested.

It can be concluded that not all the $sbf_R(t)$ greater than $msbf_\tau(t)$ or $gsbf_\tau(t)$ will ensure the schedulability of $\{\tau, R\}$ but, in the specific above-mentioned conditions, it will.

3.6 Experimental results

The main purpose of this section is to establish a relation between any $sbf_R(t)$ and the $msbf_\tau(t)$ or $gsbf_\tau(t)$, to ensure the schedulability of τ in R. What was demonstrated in previous sections is that, if $sbf_R(t)$ is equal to or grows at the same time as $msbf_\tau(t)$ or $gsbf_\tau(t)$, τ will be schedulable in R, as proven in previous sections.

Our first hypothesis consisted in the fact that any $sbf_R(t)$ function greater than $msbf_\tau(t)$ or $gsbf_\tau(t)$ would guarantee the feasibility of τ in R. In Section 3.4, this theory has been rejected. Moreover, in that section, we differentiated the three zones in accordance with the position of $sbf_R(t)$ in relation to $msbf_\tau(t)$ and $gsbf_\tau(t)$ (Figure 3.11).

In this section, the percentage of schedulable task sets according to the zone where $sbf_R(t)$ is located is calculated. To this end, a simulator that generates and evaluates different hierarchical systems is developed. It consists of the following operations:

a) Generation of the temporal model. The first step consists of generating the temporal model τ from the utilization system, U ($0 \leq U \leq 1$), and the number of tasks nT that belong to that system. Algorithm UUnifast [**?**] requires as input the number of tasks nT and the total system utilization U and provides the utilization of each task. The UUnifast algorithm is used for generating random task utilizations. This efficient algorithm allows variable independence and generates unbiased utilization values. For these reasons, it has been widely used for researchers since 2005.

b) Generation of $gsbf_\tau(t)$ and $msbf_\tau(t)$ functions from previous definitions.

c) Generation of different sequences of slots $sbf_R(t)$ and execute the system in the time given by these slots. Generation of the temporal plan.

Other useful parameters calculated are the ICI, maximum idle interval in $gsbf_\tau(t)$ and $msbf_\tau(t)$, relation between the growth of $sbf_R(t)$ and the growth of $gsbf_\tau(t)$ and $msbf_\tau(t)$, the position of $sbf_R(t)$ with regard to $gsbf_\tau(t)$ and $msbf_\tau(t)$, etc.

Moreover, previous steps can also be applied to a specific tasks set, as the one previously defined in Table 3.7. In order to make reading easier, the task set is rewritten below:

	s_i	e_i
I_1	0	5
I_2	7	25
I_3	29	30

Then, two situations are simulated:

- For the task set in Table 3.7, we generate different random sequences of slots, that is, $sbf_R(t)$, and we check the schedulability of the task set in R. The $sbf_R(t)$ generated is totally random so that the function generated can be contained in any of the zones mentioned before.

  After 300000 random sets generated, the results are as follows:

  - In cases where $sbf_R(t) < msbf_\tau(t)$, τ is never schedulable in R, as stated in Theorem 3.5.1.
  - In cases where $msbf_\tau(t) \leq sbf_R(t) < gsbf_\tau(t)$, the percentage of task set τ schedulable in R is 76.30%.
  - In cases where $sbf_R(t) > gsbf_\tau(t)$, the percentage of task set τ schedulable in R is 72.45%.

  On the basis of the results, it is deduced that although percentages are quite similar, the greater the $sbf_R(t)$, the more probability that τ will be non-schedulable. In other words, supplying more computation time than the time defined by $msbf_\tau(t)$ or $gsbf_\tau(t)$ does not imply that the task set is schedulable.

- We generate *n* task sets for each utilization factor U_t in $[0.4, 0.9]$, with $\triangle U_t = 0.1$. With $n = 300000$ iterations, the results are as follows:

  - In cases where $msbf_\tau(t) \leq sbf_R(t) < gsbf_\tau(t)$, the percentage of task set τ schedulable for each utilization factor is depicted in Figure 3.25.
  - In cases where $sbf_R(t) > gsbf_\tau(t)$, the percentage of schedulable task set τ for each utilization factor is depicted in Figure 3.26.

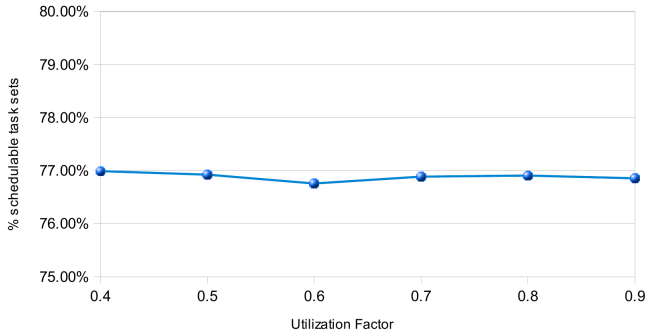


Figure 3.25: % schedulable τ in $msbf_\tau(t) \leq sbf_R(t) < gsbf_\tau(t)$

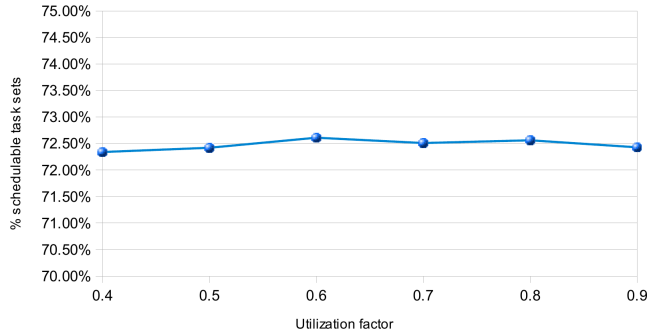


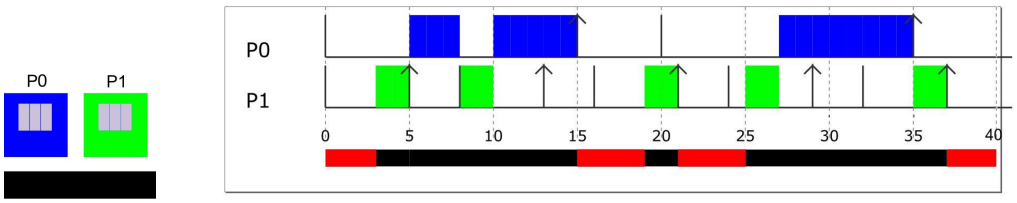
Figure 3.26: % schedulable τ in $sbf_R(t) > gsbf_\tau(t)$

According to the results, we can deduce that the utilisation factor is independent of the feasibility of the proposed method; that is, the fraction of time that the processor is busy does not contribute to a better or worse schedulability. And, moreover, as shown in the previous situation, the longer the $sbf_R(t)$, the more probability that τ will be non-schedulable (5% approximately).

3.6.1 Utilities of msbf$_\tau$(t) applied to partitioned systems

As already mentioned at the beginning of this chapter, there are some situations that make necessary the arbitrary assignment of time to a partition in the global level. An example is the addition of a new partition in a partitioned system. This new partition must match the global idle time and be schedulable, with the purpose of not recertificating the overall system, but only this new partition.

For example, 3.27 shows a partitioned system composed of two partitions and the time (in black) that the SI has to serve to the system to be schedulable. If a new partition P_2 has to be added (Figure 3.28), there is no need to re-certificate the overall system, but only the new partition P_2. Thus, the P_2 has to be scheduled in the idle time (in red) of the existing system.



(a) Partitioned system

(b) Global scheduling plan

Figure 3.27: Original allocation (a). Busy time in black and idle time in red (b)

From Figure 3.27(b) it is easy to identify the idle time available for the new partition. It is calculated in the form of ranges in Table 3.12.

Assuming that the intervals defined in Table 3.12 define a supply bound function, $sbf_R(t)$, it is possible to generate a task set whose $msbf_\tau(t)$ coincides with $sbf_R(t)$.

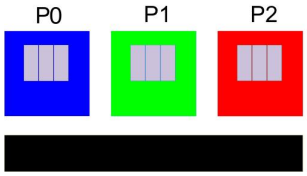


Figure 3.28: Addition of a new partition (in red)

Table 3.12: Idle intervals from Figure 3.27(b)

	s_i	e_i
I_1	0	3
I_2	15	19
I_3	21	25
I_4	37	40

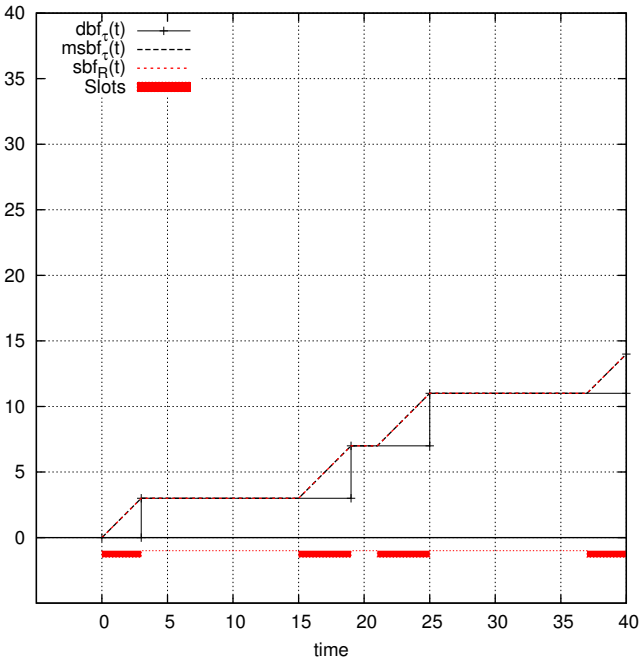


Figure 3.29: Generation of $msbf_\tau(t)$ from $sbf_R(t)$

If $msbf_\tau(t) = sbf_R(t)$, a simple $dbf_\tau(t)$ can be calculated and, consequently, a task set that accomplish these temporal parameters. A simple task set that satisfies the deduced $dbf_\tau(t)$ could be the one defined in Table 3.13. As is seen in Figure 3.29, the time slots that this task set requires to being executed coincides with the idle time slots in Figure 3.27.

Table 3.13: Task parameters τ''

	C_i	D_i	T_i
τ''_0	3	3	40
τ''_1	4	19	40
τ''_2	4	25	40
τ''_3	3	40	40

In this way, from $msbf_\tau(t)$ it is possible to calculate a task set that, allocated in a new partition, satisfies temporal requirements of an existing system.

3.6.2 Comparison of the minimum CPU supply function with other similar methods

As stated in the introduction, there are no works related to the analysis of arbitrary global CPU. Nevertheless, our methods can be compared with other works that assume a periodic supply at this level, because our model could be applied in the case of any known existing server in the global level. Specifically, [**?**] presents a schedulability analysis for a periodic supply (R=(θ, π)) at global level ($D_i = P_i \quad \forall i$) and EDF at local level with the following condition:

$$\forall 0 < t < lcm_\tau \quad dbf_\tau(t) \leq sbf_R(t)$$

The reader is reminded that a periodic supply (R=(θ, π)) provides θ units of time each π units. For more information, review Section 3.2.1.

In this paper, to find out whether a specific periodic supply R successfully schedules a set of tasks, the above equation must be simulated to obtain a solution space for θ and π. Shin and Lee [**?**] solved this equation for the following set of tasks 3.14:

Table 3.14: Task parameters τ'''

	C_i	D_i	T_i
τ_1'''	7	50	50
τ_2'''	9	75	75

In their work, Shin and Lee proposed a method to find the smallest when θ for any given resourceπ. For example, when $\pi = 10$, the minimum θ that guarantees the schedulability of the task set was 2.8.

In this section, the minimum θ using our $msbf_\tau(t)$ is obtained. Using the same task set defined in Table 3.14, the characteristic points are $t_1 = 50$, $t_2 = 75$, $t_3 = 100$ and $t_4 = 150$. At these points the values of $msbf_{\tau'''}(t)$ are: $msbf_{\tau'''}(t_1) = 7$, $msbf_{\tau'''}(t_2) = 16$, $msbf_{\tau'''}(t_3) = 23$ and $msbf_{\tau'''}(t_4) = 39$.

it has to be assured that any periodic $sbf_R(t)$ with a period of 10 must be on top of $msbf_\tau(t)$ at the characteristic points. Therefore, at t_1, 7 units of time have at least been served. It is easy to see that to fulfil this condition, at t_1 then:

$$\theta = \frac{msbf_{\tau'''}(t_1)}{\frac{t_1}{\pi}} = \frac{7}{\frac{50}{10}} = 1.4.$$

Repeating this procedure for other scheduling points and choosing the maximum θ of all, then $\theta = 2.6$. This value is lower that the one calculated by Shin and Lee, which means that their proposal is an approximation and not an exact calculation. In fact, for $\theta = 2.5$ the task set is not schedulable. Figure 3.30 shows the comparison between $msbf_{\tau'''}(t)$ and the periodic supplies with the minimum θ calculated by Shin and Lee and by us.

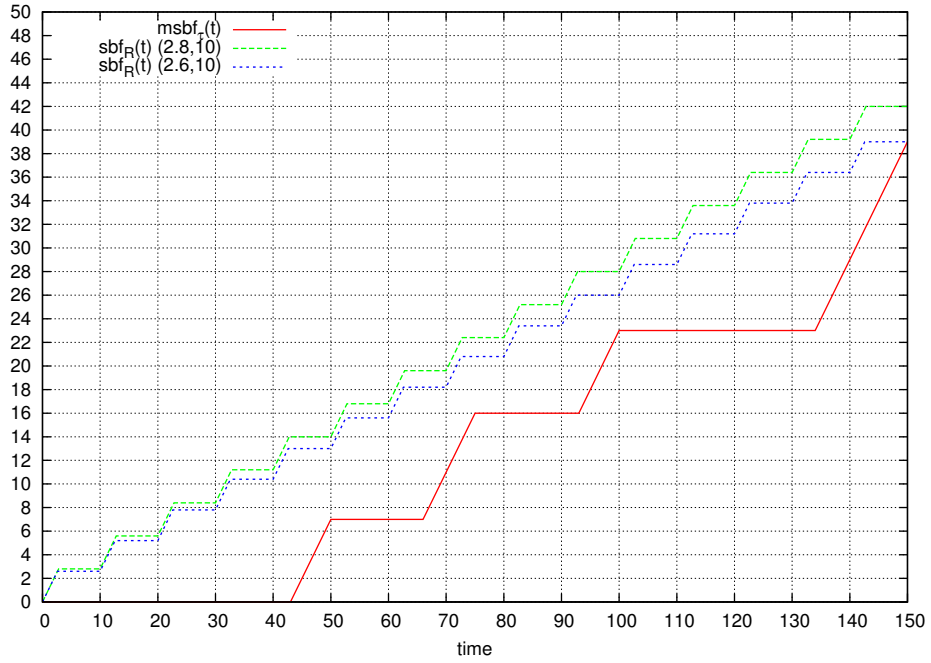


Figure 3.30: Comparison between two periodic supplies R=(2.6,10) and R=(2.8,10) and $msbf_{\tau'''}(t)$ for $\tau''' = (\tau_1'''(7,50), \tau_2'''(9,75))$

3.7 Conclusions

This section considers the case of a two level hierarchical real-time system, where local level tasks tasks are scheduled under EDF policy whereas the global level does not follow a known scheduling policy and the only information is provided as a set of CPU slots. This situation occurs in a partitioned system when CPU slots are provided by the system architect to the partition developer.

The first contribution is the calculation of CPU supply functions that ensure the schedulability of task sets; specifically, two different functions are defined: firstly, a CPU supply which offers CPU when tasks are released ($gsbf$) and, secondly, a CPU supply which offers CPU as late as possible ($msbf$), meeting the deadlines in both situations and providing the minimum CPU.

This chapter also provides a schedulability analysis of a task set, relating any CPU supply with the functions mentioned before. In other words, the proposed method can be used to check if a set of time intervals provided by the SI can be accepted by the PD or not.

Another contribution of previous functions is the possibility of using the minimum time a partitioned system needs to be completely executed in order to use its idle time to execute another partition. Thanks to this proposal, it is possible to use all CPU, exploiting the time that previously was idle in order to execute another partition. In next chapter, it will be observed that the calculus of the minimum time that a partition needs to be executed ($msbf$) offers advantages also in terms of energy savings.

Moreover, as slots are arbitrary, our model can also be used with any existing server in the global level. Thus, our model is a generalization of any scheduling in the global level.

CHAPTER

4

Energy saving techniques in partitioned systems

Energy saving has become an important issue in modern computing systems. In battery operated devices, like autonomous mobile robots, industrial controllers, wearable devices, mobile phones, etc. a lower energy consumption can lead to a longer lifetime or higher performance. Moreover, to control on-chip temperature and heat dissipation has become an important aspect in microprocessors. These factors make necessary some techniques to ensure an efficient management of power consumption. This chapter deals with energy management techniques in partitioned systems. The outline of this chapter is established as depicted in Figure 4.1.

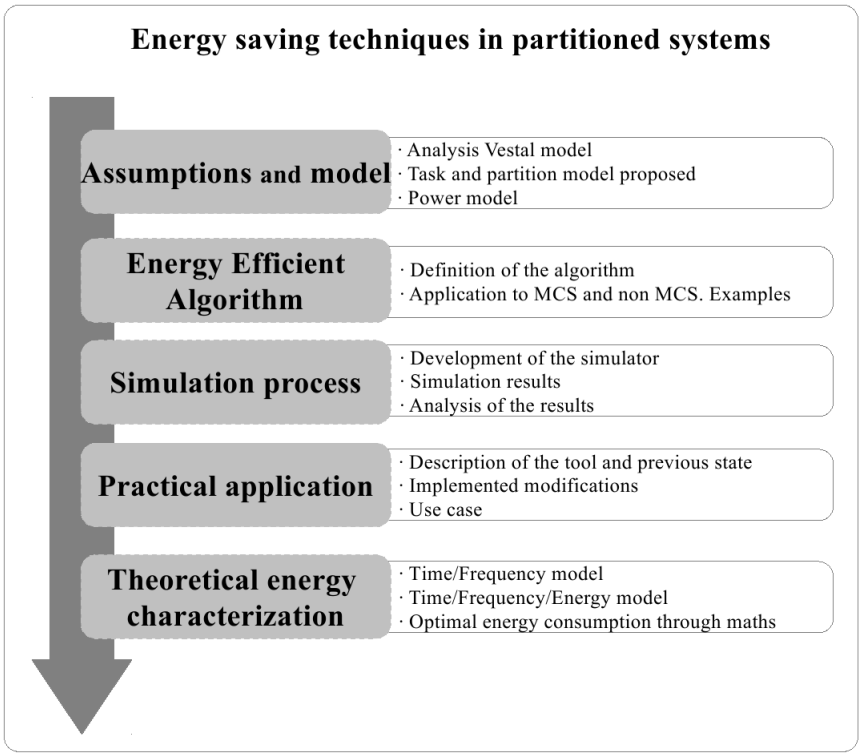


Figure 4.1: Chapter 4 layout

4.1 Introduction and objectives

Energy management is a very active research area in the recent years. In fact, energy minimization is a main requirement in the design phase of embedded systems. It will increase the reliability and decrease the heat dissipation of the system, providing a longer lifetime of battery-powered embedded systems.

In MCS context, energy management has been introduced in very few papers as stated in Section 2.6. All these works make some assumptions that do not satisfy the requirements of this work. For this reason, the objective of this chapter is to propose new energy saving strategies in the context of partitioned multicore MCS.

This chapter consists of two clearly differentiated parts: first part proposes an heuristic algorithm that saves energy in a partitioned system in which different criticality levels exist. This heuristic algorithm rests on a practical simulator, that consists of a synthetic task and partition generator and implements the energy efficient allocator algorithm and the different strategies that will be introduced later. This algorithm is also valid in a system with only one criticality level. After the simulations, these strategies have been implemented in an commercial integrated editor and analysis tool, Xoncrete, that performs the schedulability analysis of a partitioned system.

The second part solves the problem of finding the optimal energy consumption in a partitioned system from a theoretical point of view. First, a model that best fits the relation between time and frequency is defined, after evaluating the classical model existing in literature. Then, the system utilization-frequency-energy model is defined and studied in order to find the optimal energy consumption of the system.

First and foremost, as this chapter does not follow classical models as Vestal's, the proposed task and power models have to be defined. Also the typical considerations in MCS are stated in the following.

4.2 Assumptions and model

This section presents the model typically used in the design of MCS and the limitations that this model has for our requirements. Next, we discuss some methods that provide schedulability analysis in MCS with energy saving purposes and why they also fail to meet our requirements. Finally, the model that will be followed in this work is presented.

4.2.1 Vestal model and MCS state of the art

This section first presents the current model, which is typically used in mixed-criticality systems. Vestal [**?**] defined a task τ_i as an implementation of different functions and was defined as $\tau_i = \{c_i, T_i, \pi_i, C_i, D_i\}$. c_i refers to the criticality level of the task, T_i refers to the period, π_i is the task priority, C_i is the worst case execution time (WCET) of the task and D_i, its deadline. The WCET is denoted as a vector of values of WCET, in which each value corresponds to a criticality level. As usually two levels of criticality are considered (HI and LO), $C_i = (C_i^{LO}, C_i^{HI})$, being C_i^{HI} more conservative than C_i^{LO}. At runtime, the system starts the execution with LO operation mode and has to ensure deadlines for HI and LO tasks. If any HI task overruns its C_i^{LO}, the system will switch to HI operation mode and LO tasks will be dropped to guarantee deadlines of HI tasks.

This model is based on multiple execution times and this assumption is a subject of discussion because it affects the practical usability of research results [**?**]. Considering two values of C_i requires two processes to measure them: a simpler estimation process for C^{LO} and a stringent process for C^{HI}, with more pessimistic assumptions. As the certification is a crucial aspect, the acceptance of two different C_i through two different processes is a difficult issue. For example, if Certification Authorities (CAs) accepts C^{LO} for a HI critical task, there will not be need to add pessimism and vice versa. If CAs accept C^{HI} for a HI critical task, they will not accept a lower value C^{LO} that compromises the process.

Now, three state of the art mixed-criticality techniques focused on the schedulability and energy efficiency on multicore systems are discussed. These three works consider the Vestal model.

- Baruah's method [**?**]. This work first solves the partition-to-cores allocation

following the next criteria: first, the algorithm allocates HI tasks in cores using First-Fit (FF) algorithm. Then, it repeats the operation with LO tasks. It has to be checked that system utilizations per mode do not exceed 3/4. After mapping, EDF-VD [**?**] scheduling is applied to each core.

- Gu's method [**?**]. This work first uses Worst-Fit (WF) packing strategy to allocate HI tasks and tune the virtual deadlines to HI tasks in each processor. Then it uses FF to allocate LO tasks.

These two methods are focused on improving system schedulability and not on saving energy. They allow high utilization per core (i.e., using FF to maximize the system utilization in each core) and this fact leaves little idle time in order to apply, for example, DVFS techniques. On the contrary, next work is focused on exploring energy savings.

- Narayana's method [**?**]. This work studies a general energy minimization problem for MCS on multi-cores, considering also different system operation modes (HI and LO). It also introduces weight factors *w*, that represent the percentage of time a task is in HI or LO mode ($w_{HI} + w_{LO} = 1$). This works springs from previous methods, applying WF first to HI tasks and then to LO tasks, with core utilizations upper-bounded by 3/4. Then, DVFS is applied per core and EDF-VD is selected for scheduling. All this process is based on a linear search in order to find the combination of task-to-cores with the minimum energy consumption and minimum number of cores. They also provide another method based on the isolation of tasks: HI tasks are allocated to HI cores and LO tasks, to LO cores.

It is deduced from all this that these works consider that the system may change from non critical state to high criticality if any HI task trespasses its C^{LO} value. As we are not considering WCETs as multiple values per task and neither do we consider dynamic changes in the criticality of the tasks, we are not following this model and these assumptions are non-applicable in our case. Thus, there is also no point in having weight factors because each task is executed in only one mode (with one WCET), so the probability of being executed in that mode is always 1.

As a reminder, this work considers a partition as a set of tasks being the partition the unit that is allocated into a core. The overall system is a mixed-criticality partitioned system on multi-core, based on the Xtratum hypervisor (See Figure 4.2).

XtratuM hypervisor (See Section 2.7) requires a static configuration file that includes, among others, the temporal requirements of the system. These are given as a set of time slots per core, each of them characterized by its start time, duration and the identifier of the partition that will be executed in this slot. For this reason, all works that consider dynamic changes during execution time do not completely meet our requirements. Neither the works that focus on energy management based on dynamic techniques are suitable in our context.

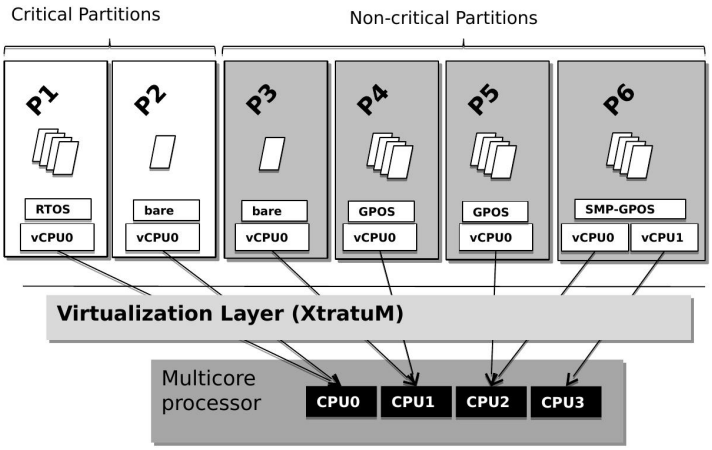


Figure 4.2: General overview.

All previous requirements are imposed by the applications used in the avionics and railway sector in order to meet ARINC 653 standard [**?**], since the results of this research will be applied in H2020 project SAFEPOWER [**?**]. This project's goal is to enable the development of low power mixed-criticality systems through the provision of a reference architecture, platforms and tools to facilitate the development, testing, and validation of these kinds of systems according to the market needs. In these experiments, intensive simulations have been performed to obtain a large number of computation time measurements. Details about these experiments can be found in [**?**].

Now, we are going to define the model that best fits our requirements.

4.2.2 Task model proposed

We consider a set of m heterogeneous cores $M_1..M_m$. Each core can execute with any g frequencies independently, within the range $[f_1, f_g]$, being f_g the highest frequency of each core. As cores are heterogeneous, each of them works in a different range of frequencies. However, for simplicity and to be able to adapt our model to SAFEPOWER project requirements, we suppose that all cores work in the same range of frequencies.

A set of p partitions $P_1..P_p$ are statically allocated to the cores. Each partition P_i is defined by the pair $P_i = (\tau, \text{L})$, where τ is the set of tasks and L is the criticality level of the partition.

We define two criticality levels per partition [**?**]: HI (high) and LO (low). All tasks that belong to the same partition have the same criticality level. HI partitions have to be executed to completion in any condition and temporal constraints of their tasks have to be fulfilled. LO partitions can be executed in some conditions, depending on the impact caused in the system for not being executed. No temporal constraints are identified but it is expected a bandwidth for them. If, in some cases, they can be dropped, we call them disposable LO partitions (DLO). If they have to be executed in any case they are called required LO partitions (RLO). But, even if these partitions cannot be dropped, their bandwidth can be reduced if needed (for energy saving purposes). Dropping and other operations will be introduced later.

Each partition P_i is composed of n_i tasks $\tau_i = (\tau_{i1}..\tau_{in_i})$. Each task τ_{ij} is characterized depending on its criticality:

- HI tasks are defined as $\tau_{ij} = (C_{ij}, D_{ij}, T_{ij})$ where T_{ij} is the period, D_{ij} is the deadline and it is supposed to be equal to T_{ij} and C_{ij} is the active WCET. Active WCET of a task depends on the running processor frequency. Thus, computation time of each task is denoted as an array $[C_{ij}^{f_1}, .., C_{ij}^{f_g}]$ in which $C_{ij}^{f_i}$ is the WCET estimated at frequency f_i. A common assumption is that both computation time and utilization change linearly with respect to the inverse of the frequency ([**?**],[**?**]). This is only a simplification of the problem.

  There is no need that all tasks spend their WCETs in all executions. Thus, we consider the WCET to do the analysis and it will ensure that any other value will be supported by our study. In order to simplify the problem of shared

resources among cores, we assume that the interference between cores is treated, in terms of time, as a part of the WCET of each task. In recent works, WCET is considered not to be fully scalable with the processor speed [**?**] [**?**], because there are several operations that do not share the clock frequency with the CPU. In these situations, WCET can be split in a fixed and another variable portion.

The utilization of a task τ_{ij} running at frequency f_i is $U_{ij}^{f_i} = \frac{C_{ij}^{f_i}}{T_{ij}}$.

- LO tasks τ_{ij} are characterized by their bandwidth $\eta_{ij} = \frac{B_{ij}^{f_i}}{T_{ij}}$, where $B_{ij}^{f_i}$ is the budget or WCET at frequency f_i and T_{ij} is the period of a task τ_{ij}. The utilization $U_{ij}^{f_i}$ of a task τ_{ij} running at frequency f_i is η_{ij}. In this sense, we call equally bandwidth or utilization to the relation between computation time and period. When a task is characterized by a bandwidth, what is known is the relation between the WCET and period, but not their explicit values. There are a lot of WCET-T combinations that provide the same bandwidth.

The computation time C_{P_i} of a partition P_i is the sum of the computation times of the tasks that belong to that partition. Equally, the utilization of a partition is the sum of the utilizations of all tasks that belong to that partition.

We may assume without loss of generality that all preemptions occur at integer time values. We then assume, for the remainder of the work, that all parameters are indeed integers. Denote that the internal partition context switch in taken into account in the task computation. Moreover, core migration of a partition or a task is not allowed.

Note that, as stated before, from the criticality point of view, our model is simpler than Vestal's model [**?**] in the sense that we assume the longest execution time observed in testing as a unique WCET estimate for each frequency. The reason is to avoid having a computation times matrix (two dimensions: for frequency and criticality level). Vestal's model also use the same term "criticality" to refer both to the criticality of a task and the mode of operation [**?**]. We consider that the system has different operational modes each one associated a static schedule. In Vestal's model, two different operational modes will differ only in the computation times. In our model (based on ARINC-653 Part 2 [**?**]), each operational mode can have associated a different set of tasks.

Regarding to the distribution of tasks and partitions to cores, methods presented in this paper are able to solve two situations:

- On one hand, if all tasks belonging to the same partition are assigned to the same core, the unit which is allocated to cores will be the partition.

- On the other hand, if all tasks of the same partition are assigned to different cores, the allocation unit is the task.

Figure 4.3 shows in A the first situation, in which the different tasks of the same partition are allocated to different cores. In B, all the tasks that belong to the same partition are allocated to the same core. This is the same as saying that the partition is allocated to the core.

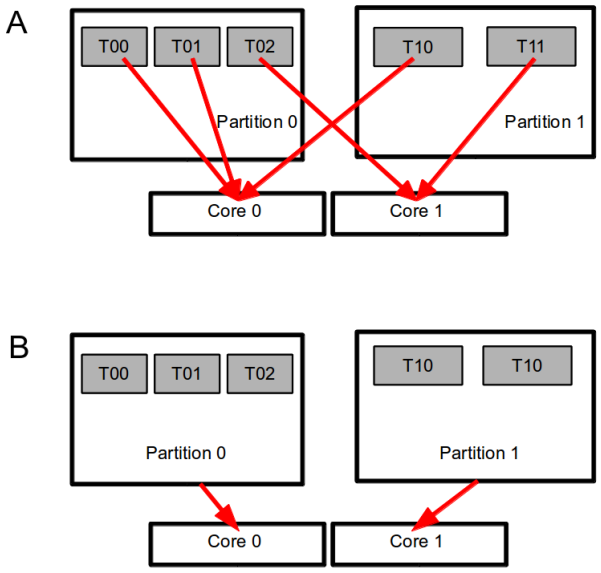


Figure 4.3: Tasks or partitions allocated to cores. A: Tasks. B: Partitions.

Henceforth we will consider that all tasks in a partition are assigned to the same core in order to avoid overheads. Thus, from now on, partitions will be allocated to cores and the total core utilization U_{M_i} is defined as the sum of all utilizations of the partitions assigned to that core M_i.

4.2.3 Power model

Power consumption is divided in dynamic consumption and static consumption. The former depends on the activity of the processor while the latter is mainly due to leakage current and can only be reduced by activating a low-power state. We will assume the power model from [**?**]:

$$P(f) = P_s + \beta f^{\alpha} \tag{4.1}$$

where $P(f)$ is the total power consumption at operating frequency f, P_s is the static power consumption and βf^{α} is the dynamic power consumption. P_s is introduced in the system due to the leakage current and βf^{α} is introduced due to capacitor charging and discharging during processor activity. α is a fixed parameter determined by the hardware. A common assumption is that $2 \leq \alpha \leq 3$. β is a constant that depends on the effective switching capacity, $\beta > 0$. It is clear that the power consumption function is a convex-increasing function of the processor frequency. A common way of reducing dynamic consumption is to reduce the processor frequency through DVFS, as stated in Section 2.6.

There are some works that miss out the static power into the analysis of energy consumption. It is due to the fact that P_s is always consumed because several components in real-time embedded systems are put in low-power states for energy savings purposes but never turned off completely [**?**]. In our problem, both static and dynamic power consumption are considered. Decreasing system frequency supposes increasing the time that the processor is turned on and, thus, energy dues to the static power also increases.

4.3 Energy Efficient Partition Allocator

In next sections, we are going to present our solution to the mixed-criticality energy minimization problem (from the allocation point of view). This solution is based on the utilization of an allocation heuristic.

Allocation to cores can be solved using a bin-packing algorithm. The approach requires to define objects (partition utilizations U_i), which can then be packed on to the bins (cores).

Among all these heuristics, Worst Fit Decreasing Utilization (WFDU) is known to obtain a well-balanced load among cores. In [**?**] it is demonstrated that this balance-load also minimizes energy consumption. But this statement is done for systems with applications running at the same frequency.

In this section, an energy efficient partition allocation is presented in which each partition can run at a different frequency. The concept of mixed criticality systems (MCS) will be applied and a comparison between using non MCS (hard systems) and MCS will be studied.

4.3.1 Energy efficient partition allocation in non MCS

The starting point of this section is a motivational example in order to justify the impact of partition mapping on energy minimization.

This example is taken from [**?**] adapted for our partition model. Consider four partitions with $U_1 = 0.5$, $U_2 = 0.4$, $U_3 = 0.4$ and $U_4 = 0.3$ running on $m = 2$ heterogeneous processors. By [**?**] we know that the allocation that minimizes energy is $U_{M_1} = U_1 + U_4 = 0.8$ and $U_{M_2} = U_2 + U_3 = 0.8$ since it is the most balanced one. But this is considering that all partitions run at the same frequency. Following the model defined before, with $g = 2$ frequencies (f_1, f_2) then each partitions will have 2 possible utilizations. The values are shown in Table 4.1. Now, allocators can choose one of the two utilizations for each partition. But which one of all possible combinations results in the most energy efficient?

4.3.1.1 Energy efficient partition allocation algorithm

In this section an energy efficient partition allocation is presented in which each partition can run at a different frequency. This frequency will not change throughout

Table 4.1: Utilizations

	U_i^{f1}	U_i^{f2}
P_1	0.7	0.5
P_2	0.56	0.4
P_3	0.56	0.4
P_4	0.42	0.3

the system execution. Making an analogy with priorities, DVFS will treat frequencies as dynamic priorities while our method considers frequencies as static priorities. However, our method does not prevent any further dynamic slack reclaiming. Our proposal is not optimal, since computing the allocation that minimizes overall energy consumption is intractable.

Algorithm 1 shows the EEA (Energy Efficient Allocator) algorithm to allocate partitions to cores.

In this algorithm we can choose the type of allocator and the criterion with which the partitions are selected to decrease its frequency. As allocators we have considered Worst Fit Decreasing Utilization (WFDU), First Fit Decreasing Utilization (FFDU) and Best Fit Decreasing Utilization (BFDU) (the same ones that have been compared in [**?**]). As sorting criteria, partitions can be chosen by decreasing utilization (DU), increasing utilization (IU) or randomly (R). This way we can make a comparison between these allocations algorithms and know which of them is better for energy management purposes. The resulting algorithms and their names are shown in Table 4.2.

The algorithm starts assigning the highest frequency to all partitions (line 4). The allocation of this partition set ($k = 0$) is called the original mapping that is also the mapping with the most energy consumption (lines 2 and 3). Then, a partition is selected (following the criteria DU, IU or R) to decrease its frequency to f_{g-1} (line 9 that calls the Algorithm 2) and an allocation is again performed ($k = 1$). The algorithm runs sequentially decreasing partition frequencies (line 13) until we reach a non-feasible mapping or all the partitions have reached the minimum frequency. If any solution is not feasible at any point, the system will return to the immediately previous frequency (line 15). We consider a feasible mapping if the utilization of each core is below 1.

Algoritmo 1: Allocation for energy management

```
Data: Allocator, sorting
Result: mapping
1  Function EEA (Allocator, sorting)
2      k → 0;
3      mapping ← core workload;
4      Set all partition frequencies to the maximum frequency;
5      Loop
6          mapping(k) = Allocate (Allocator);
7          if feasible then
8              k → k+1;
9              P_i=selectPartition(sorting);
10             if P_i = −1 then
11                 exit;
12             end
13             decreaseFrequency(P_i);
14         else
15             increaseFrequency(P_i);
16             exit;
17         end
```

Algoritmo 2: Partition selection

```
   Data: sorting
   Result: Allocated partition or not
1  Function selectPartition (sorting)
2      sort partitions according to sorting criteria to create an array of
         heaps Ulist;
3      for i = g; i > 2; i = i + 1 do
4          p=Head(UList(i));
5          if p! = null then
6              return p;
7          end
8      end
9      return -1;
```

Table 4.2: Types of algorithms

Part. selection / Allocation alg.	DU	IU	R
FFDU	FFDU2	FFDUIU	FFDUR
BFDU	BFDU2	BFDUIU	BFDUR
WFDU	WFDU2	WFDUIU	WFDUR

As explained in Section 4.2.1, our model does not consider two WCETs per task as Vestal, but only one value measured in the worst case scenario. For this reason, the schedulability test is summarized in ensuring that the utilization per core does not exceed 1 rather than applying sophisticated mixed-criticality schedulability tests [**?**]. Basically, after mapping, each core can be tested as a single-core system with implicit deadlines (deadlines equal to periods) so it will be schedulable if the utilization is less than 1 [**?**].

The frequency decrease works as follows (Algorithm 2): for each frequency f there is a heap of partitions that have assigned this frequency sorted according to the sorting criteria. The partition to decrease its frequency is chosen from the head of the list of the highest frequency (line 4). If the only non-empty heap is the one with frequency equal to f_1 the function $selectPartition$ returns -1 and the

algorithm stops since all partitions have reached their lowest frequency.

The result of this algorithm is a mapping with the minimum consumption before the system becomes unfeasible. In addition, each iteration of the loop gets a mapping with less consumption than the previous one. So, we can compare bin-packing algorithms by energy consumption of the last mapping and parameter k that is the number of mappings obtained by the algorithm.

4.3.1.2 **Example**

Back to the motivational example of this section, Fig 4.4 shows the iterations of the EEA algorithm. We have run WFDU2, that is, partitions have been allocated following WFDU and the partition selection criteria is by decreasing utilization. Fig 4.4A corresponds to the original mapping, in which all partitions run at the maximum frequency f_2. In the first iteration, P1 is selected to decrease its frequency, so we change its utilization from $U_1^{f2} = 0.5$ to $U_1^{f1} = 0.7$. The result is shown in Fig 4.4B. In the next iteration (k=2), P2 is selected to decrease its frequency resulting in Fig 4.4C. The algorithm stops because the next mapping will cause $U_{M_2} > 1$.

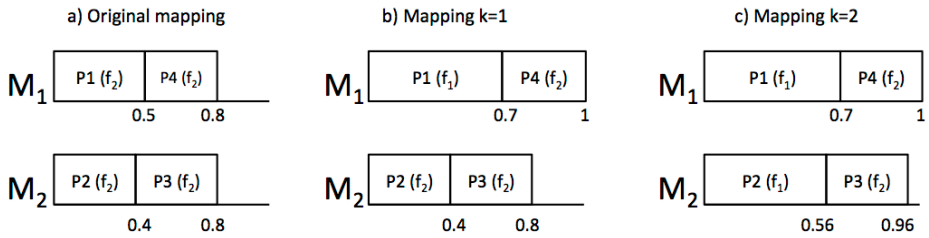


Figure 4.4: EEA mappings

For the calculation of the energy consumption, we assume $\alpha = 3$, $\beta = 1$ and $P_s = 0.8W$ [**?**], [**?**]. The processor frequencies are $[f_1, f_2] = [0.8, 1.1]GHz$. The energy consumption for an hyperperiod H is calculated as $E = P(f)H$. The results are shown in Table 4.3. As we see, increasing the total utilization of the cores by decreasing the frequencies at which partitions run results in an energy saving with respect to the original mapping up to a 8% in this simple example.

Table 4.3: Energy consumptions (Ws) of the mappings of the example

	M_1 (Ws)	M_2 (Ws)	Total (Ws)
Mapping a)	3.4096	3.4096	6.8192
Mapping b)	3.1154	3.4096	6.525
Mapping c)	3.1154	3.1742	6.2896

4.3.2 Energy efficient partition allocation in MCS

Once the energy efficient allocation method has been presented, in this section we are going to complete the proposal with the inclusion of mixed-criticality systems requirements.

In the previous section all partitions have the same criticality level, but when partitions have different levels of criticality, we can follow the next strategies to save energy:

- Trim the bandwidth of RLO partitions. RLO partitions cannot be dropped but their bandwidth can be reduced. We propose the reduction in the following way:

  For a RLO partition P_i with $U_i^{f_1}$,..,$U_i^{f_g}$, it is trimmed when it executes at the lowest frequency f_1 but with the utilization of the highest frequency f_g. In practice, this is possible thanks to the utilization of a static cyclic scheduler in which we can force the duration of the slots. This way we can assign $C_{ij}^{f_g}$ units of time but running at frequency f_1.

  Let's assume that for a time window of length the hyperperiod H, the units of time that a partition P_i executes under frequency f_1 is:

  $$H\frac{C_i^{f_1}}{T_i} = HU_i^{f_1} \tag{4.2}$$

  When a partition is trimmed, it executes

  $$H\frac{C_i^{f_g}}{T_i} = HU_i^{f_g} \tag{4.3}$$

  Therefore, there is a performance loss of:

$$1 - \frac{U_i^{f_g}}{U_i^{f_1}} = 1 - \sum_{j=1}^{j=n_i} \frac{C_{ij}^{f_g}}{C_{ij}^{f_1}} \tag{4.4}$$

- Trim the bandwidth of DLO partitions. This is done in the same way as for RLO partitions. The performance loss will follow the same equation.
- Drop DLO partitions. As stated in previous sections, these partitions can be dropped if needed. In this case, the performance loss of this partition is 1.

To summarize, dropping a partition means not running it, while trimming is an intermediate solution between executing completely the partition and not executing it at all. A trimmed partition executes at its lower frequency but with the highest WCET. It is indicated for partitions whose activities are "best effort", that is, partitions with soft timing constrains, as they are a LO-criticality partitions.

Depending on how we combine these strategies, we will obtain different mappings with different grades of energy consumption and performance. The idea is to trim and/or to drop RLO/DLO partitions and execute EEA algorithm, that will use the free utilization of DLO and/or RLO partitions to decrease frequencies as much as possible. Then, we are going to define the profiles, taking into account that each profile is a consequence of the previous profile, i.e., adds changes to the previous actions. We propose the following profiles:

- Profile 1: Maximum energy saving without performance loss. In this case, we do not allow to drop any DLO partition neither trimming RLO partitions. Therefore, this is equivalent to treat all RLO and HI partitions as HI partitions and the EEA algorithm is executed as explained before.
- Profile 2: This profile will only allow trimming DLO partitions.
- Profile 3: This profile will also allow trimming RLO partitions.
- Profile 4: In this profile we drop all DLO partitions.
- Profile 5: In this profile we execute the EEA algorithm in HI partitions as much as possible.

It is clear that as long as the number of the profile increases, the energy saving and the performance loss also increases.

These profiles are different mappings in which each partition will run at a specific frequency. After packing, a schedulability test must be applied. As we mentioned in initial sections, our model does not consider different criticality modes per task and, thus, tasks only have one level of WCET fixed before the execution (see Section 4.2.1). For this reason, the schedulability test is summarized in ensuring that the utilization per core does not exceed 1 rather than applying sophisticated mixed-criticality schedulability tests [**?**]. Basically, after mapping, each core can be tested as a single-core system with implicit deadlines (deadlines equal to periods) so it will be schedulable if the utilization is less than 1 [**?**].

Then, each profile will derive into a static scheduling plan stored in the configuration file of the hypervisor. This way, 5 profiles will be used to define every single scheduling plan. At runtime, a supervisor partition will be in charge of change the profile depending on the energy state of the system. This change always occurs when the MAF is completed, so every task completes its execution. An example of the advantage of changing plans is to link them to the battery level. For example, Profile 1 will operate when battery level is between 100-81%, Profile 2 with 80%-61%, Profile 3 with 60%-41%, Profile 4 with 40%-21% and Profile 5 with 20%-1%.

We would like to specify that, although this section is generalized to multicore systems, the previous strategies may be applied to single core systems. In this scenario, EEA algorithm will offer always the same solution, due to the impossibility of allocating different partitions in different cores (there is only one core). However, trimming and dropping partitions is possible and, thanks to this operations, the total energy consumption will be reduce.

4.3.2.1 **Example**

Using the same example as before, now we will assume that P_1 and P_2 are HI partitions, while P_3 is RLO and P_4 is a DLO partition. Profile 1 is equivalent to the mapping obtained in Fig 4.4C.

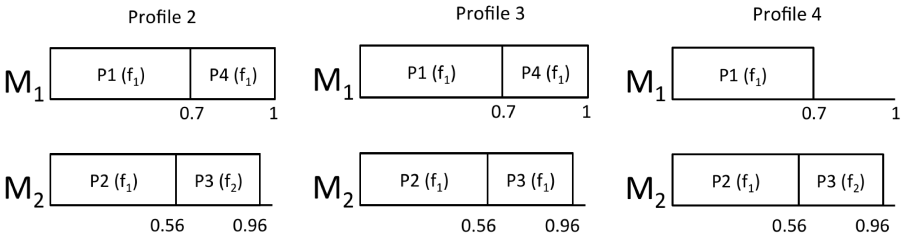
From this starting point, we can derive the rest of the profiles:

- Profile 2: In this profile, the utilization of P_4 for f_1 is trimmed from 0.42 to

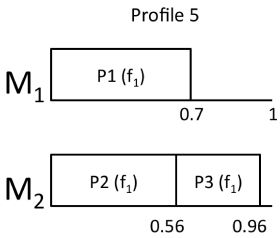
0.3.

- Profile 3: In this profile, the utilization of P_4 for f_1 is trimmed from 0.42 to 0.3 and the utilization of P_3 for f_1 is trimmed from 0.56 to 0.4.
- Profile 4: This profile does not contain P_4.
- Profile 5: In this profile, P_4 does not exist and the utilization of P_3 for f_1 is trimmed from 0.56 to 0.4.

Fig 4.5 shows the resulting mappings of the previous profiles. In Profiles 4 and 5 (subfigures (c) and (d)), dropping partition P_4 does not achieve any benefit because the extra "space" cannot be used to allocate any more partition in M_1. The different energy consumptions are presented in Table 4.4.



(a) Mapping for Profile 2 (b) Mapping for Profile 3 (c) Mapping for Profile 4



(d) Mapping for Profile 5

Figure 4.5: Mappings depending on the profile.

This performance loss is 0.25 for P_4 trimming, 1 for P_4 dropping. Regarding P_3, performance loss is 0.28. We can see how this time, the total energy saving (24.87%) is more important at the expense of losing performance.

Table 4.4: Energy consumptions (Ws) of the different profiles

	M_1 (Ws)	M_2 (Ws)	Total (Ws)
Profile 2	2.624	3.1742	5.7982
Profile 3	2.624	2.5190	5.143
Profile 4	1.8368	3.1742	5.011
Profile 5	1.8368	2.5190	4.3558

4.4 Simulation process

We developed a simulator to implement the proposed algorithms and a synthetic task and partition generator.

A number of tests have been run, specifically 10^5 synthetic partition sets have been generated for $m = 4$ and total utilizations varying from 2.5 to 4 in steps of 0.1, resulting in 1500000 total simulations.

The synthetic task and partition generator is developed as follows:

- The maximum total load is defined by the user or calculated by the simulator in relation to the number of cores m.
- Once the maximum load is determined, it is distributed among cores, in the knowledge that 100% is the maximum utilization per core. Once the utilization per core is established, the sum of all utilizations of the partitions allocated in that core should not exceed this value.
- Now, the number of partitions for each criticality level is calculated as follows:
  - HI level: a random value within the interval [4,8].
  - LO level (RLO): a random value within the interval [3,6].
  - LO level (DLO): a random value within the interval [3,8].
- Then, a maximum total load per criticality level *L* is calculated.
- Partitions utilizations are generated using the UUniFast-discard algorithm [**?**] that gets an unbiased distribution of the maximum load per criticality U^L_{total} among the partitions of that criticality level. This will accomplish that $U^L_{total} = \sum_{\forall P_i \epsilon L} U_{P_i}$. This algorithm is an improvement of the UUnifast algorithm [**?**], that discards a task set if the utilization of any of its tasks is greater than 1.
- Regarding to tasks, the number of tasks per partition is calculated as follows:
  - HI partitions: a random value within [2,8] tasks per partition.
  - LO level partitions (RLO): one task per partition.

  - LO level partitions (DLO): one task per partition.

- Tasks utilizations were also generated using the UUniFast algorithm. Tasks parameters are calculated for the greatest frequency, f_g. Once task utilization has been deduced, period is selected *randomly* in such a way that the hyper-period of tasks is not a very big value. Then, the computation time of τ_{ij} at a frequency f_g is calculated as $C_{ij}^{f_g} = U_{ij}^{f_g} \cdot T_{ij}$.

Computation time $C_{ij}^{f_i}$ of a task τ_{ij} increases by reducing its frequency f_i, i.e. the higher the frequency, the shorter the computing time. To simplify the problem, we suppose that this relation is linear, as in [**?**] and [**?**], in spite that the fact that [**?**] shows that this relation best fits a rational function. In particular, in the first step, computation times of tasks are generated with the greatest frequency. Consequently, to complete the array of values of WCET, we use the values of available system frequencies as follows:

Definition 4.4.1. Let us denote as f_p the value from which frequencies are greater or equal to 1 within the range $[f_1, ..., f_p, ..., f_g]$. Then:

$$C_{ij}^{f_{m-1}} = \begin{cases} \frac{C_{ij}^{f_m}}{f_m} & \text{if } 0 \leq m \leq p \\ C_{ij}^{f_m} \cdot f_m & \text{if } p+1 \leq m \leq g \end{cases} \tag{4.5}$$

All these parameters are saved in an array and used for frequency changes in tasks. It is clear that linear relation is a simplification of the problem [**?**]. If this relation is not known, the user will provide the array of computation times to make the work succeeds.

When everything is ready, we start the simulator to calculate different mappings with different grades of energy and performance, as was explained before.

We have conducted the same experiments for 8 cores. In this case, the number of partitions have been multiplied by 2. Other experiments consist on using 2 cores. In both cases, similar results have been obtained as for 4 cores.

4.4.1 Comparison of allocation methods in non MCS

In this section a comparison between the different allocators and partition selection is done. We measure two parameters:

- Energy saving: This is the saving of the final mapping with respect to the original mapping, that is, the mapping at which all partitions run at the highest frequency (iteration $k = 0$).

- Number of feasible mappings. This corresponds to the iterator k of Algorithm 1.

Figs 4.6A, 4.7A and 4.8A show the results for 4 cores in terms of energy savings. In this pictures, the relation between energy saving and utilization factor (from 240% to 400% in the case of 4 cores) is depicted. As utilization factor increases, it is observed that energy saving is reduced. When cores are almost full (utilization 380-400%), reducing frequency (i.e. increasing computation times) will make the system infeasible. For this reason, the scope of energy saving is short.

As it is seen in the figures, the three base bin packing algorithms present very similar results. This is not a surprise, since we are measuring the energy of the final mapping, which corresponds to a situation in which all the cores will be at full capacity. Moreover, there is also no difference in the partition selection criteria as far as energy saving is concerned.

However, the number of mappings is depicted in Fig 4.6B, 4.7B and 4.8B. It seems that BFDU is the algorithm that needs more iterations to reach the optimal solution and R is clearly the worst partition selection criteria, due to the randomness of its results. It is clear that the more utilization factor, the less iterations are possible to perform.

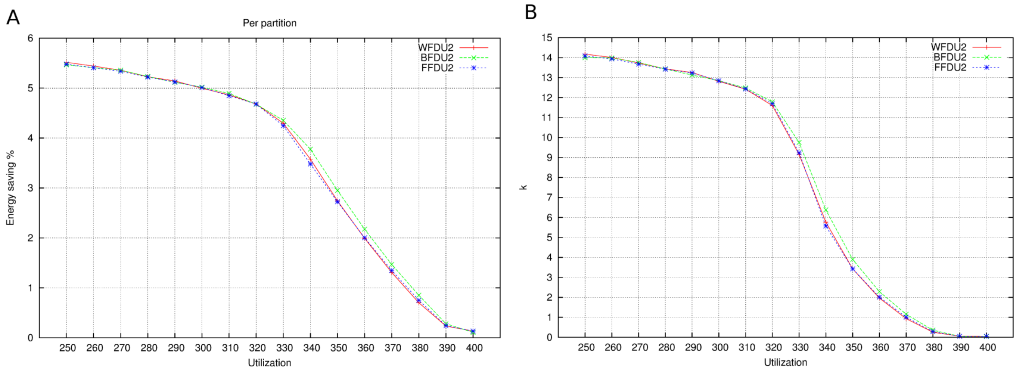


Figure 4.6: DU. A: Energy saving. B: Number of mappings.

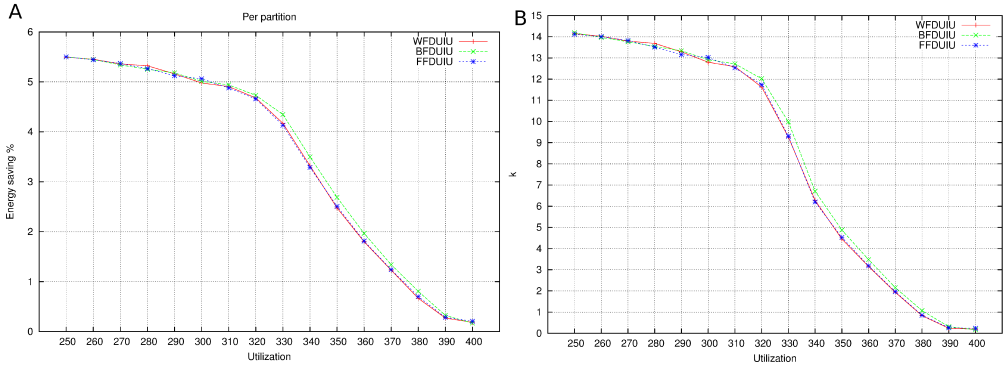


Figure 4.7: IU. A: Energy saving. B: Number of mappings.

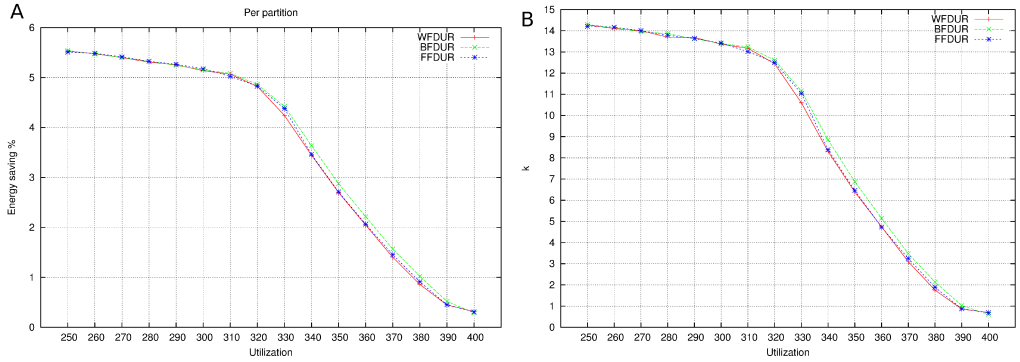


Figure 4.8: R. A: Energy saving. B: Number of mappings.

To avoid adding all the results in this paper, we show in Fig 4.9 the energy savings when FFDU is selected as allocator. Although we know that different allocators provide very similar results, FFDUIU allocator provides slightly worst outcomes.

Fig 4.10 depicts, as in previous figures, the relation between energy saving and utilization factor but, in this case, experiments have been developed in 2 (4.10A) and 8 cores (4.10B). It is demonstrated again that energy saving decreases with utilization factor, being almost zero when cores are getting full.

As different allocators provide similar results, let us complement the results section with a comparison between an exact method (constraint programming solutions, CP) and our heuristic. In almost all situations CP provides the same solution as our heuristic. But, in some scenarios, CP provides better energy savings

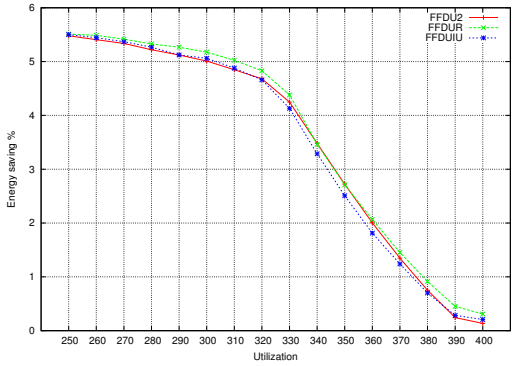


Figure 4.9: Energy saving no MCS when allocator is FFDU

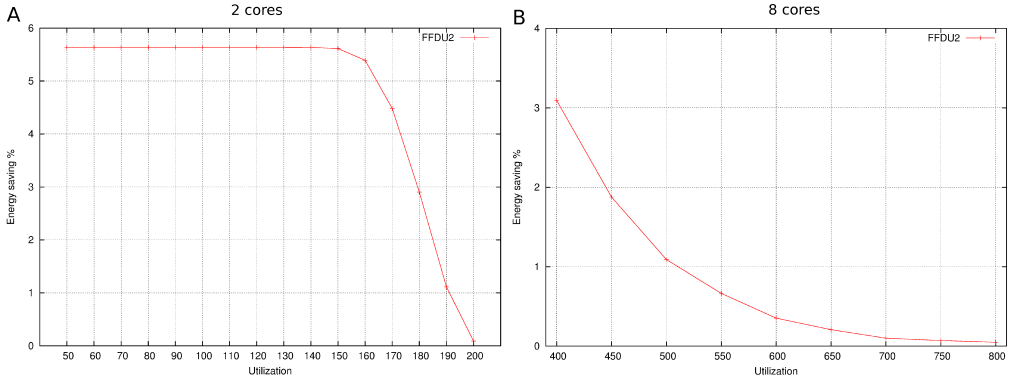


Figure 4.10: Energy saving with 2 and 8 cores. A: 2 cores. B: 8 cores.

than our algorithm. However, in terms of time consumptions, our algorithm offers much better results.

Figure 4.11 depicts the average time used by the CPU in executing 50 iterations of the EEA algorithm. Each iteration consists of allocating a number between 10 and 20 partitions in 4 different cores, in order to minimize the energy consumption. The time measured in CP simulator is directly provided by the solver. The time in heuristic algorithm is calculated measuring the number of instructions and the frequency of the CPU. We can obverse that the more the system utilization is, the less time the algorithm needs to reduce the system frequency (increasing system utilization is less possible).

We observe that only one experiment expends between 15 and 30 minutes and even more, depending on the system parameters, number of cores and partitions, number of available frequencies, etc. In a simple situation with 4 partitions allo-

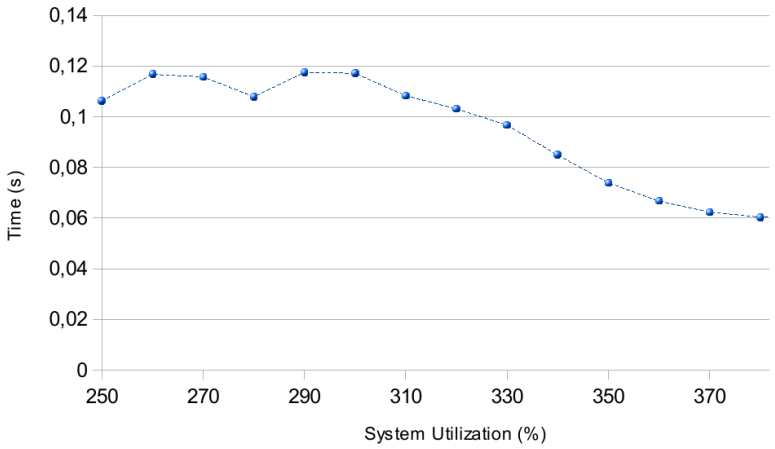


Figure 4.11: Time consumption in executing EEA algorithm.

cated in 2 cores as in Table 4.5 with 2 system frequencies, the times used to solve the problem in each situation and energy savings are summarised in Table 4.6.

Table 4.5: Utilizations - Example

	U_i^{f1}	U_i^{f2}
P_1	0.62	0.5
P_2	0.41	0.35
P_3	0.69	0.58
P_4	0.45	0.37

Table 4.6: Comparison between EEA and CP solvers.

	E_{M_1} (Ws)	E_{M_2} (Ws)	TOTAL ENERGY (Ws)	Simulation Time
EEA	10.4275	14.725	25.1525	0m0.202s
CP	12.6225	12.1125	24.735	14m 5s

In Table 4.6, it is observed that the energy consumption is bigger with EEA than CP, but the time the CP solver needs to find the solution is significantly higher.

In next subsection we use the heuristic simulator in order to evaluate the situation in a mixed criticality system.

4.4.2 Energy saving and performance loss in MCS

We conducted the same set of experiments to measure the energy saving achieved, the performance loss and number of mappings of the 5 profiles explained previously.

Figure 4.12 depicts energy savings with different allocators and profiles. It shows the more system utilization increases the more energy saving decreases. If cores are almost full, decreasing system frequency is becoming increasingly difficult and saving energy is also difficult.

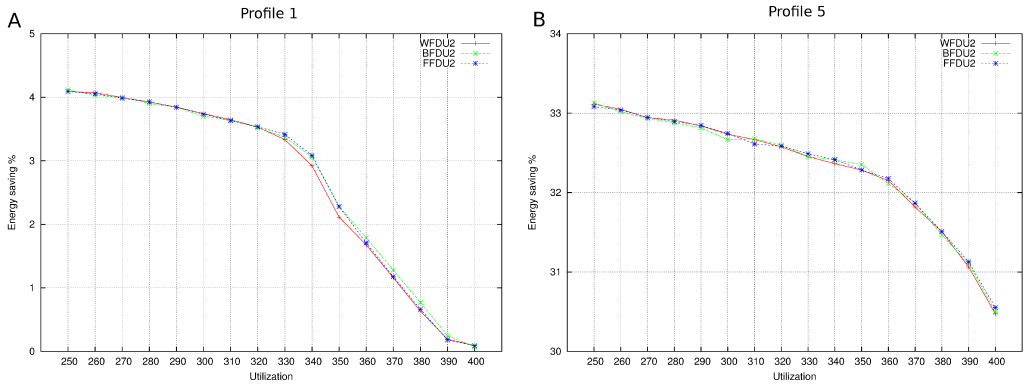


Figure 4.12: Energy saving with different allocators. A: Profile 1. B: Profile 5.

WF bin-packing balances the total load between cores but FF is considered as the strategy that performs better than others in the performance of partitioned scheduling [**?**]. As seen in Figure 4.12, FFDU presents slightly better results than others (as in non MCS) and Figure 4.13 shows the different variations of FFDU algorithm in the profile with the maximum energy savings.

With FFDU2, we measure the parameters mentioned before: energy saving, performance loss and number of mappings.

- Energy saving is measured as in non MCS. As each profile uses lower system frequency than the previous profile, energy saving will be greater as profile increases. It is depicted in Fig 4.14. When DLO tasks are dropped and the system frequency is reduced as much is possible (profile 5), energy saving is about 35%.

- Performance loss in profile i is calculated as the relation between the system execution times in profile i and profile 0, being profile 0 the original system

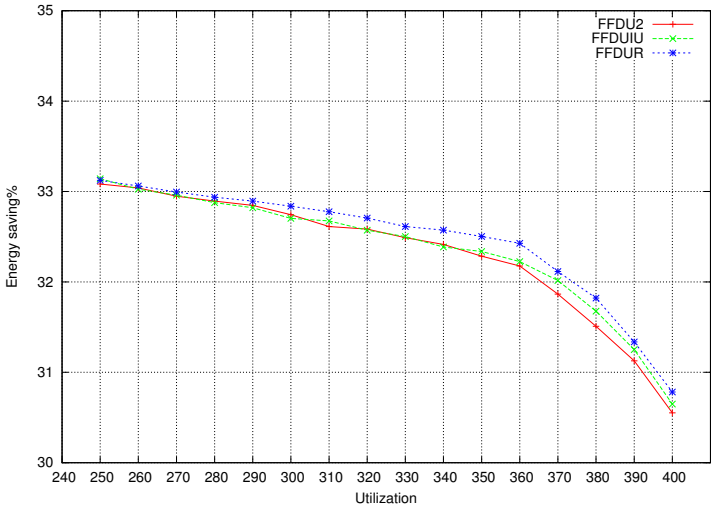


Figure 4.13: Energy saving MCS when allocator is FFDU

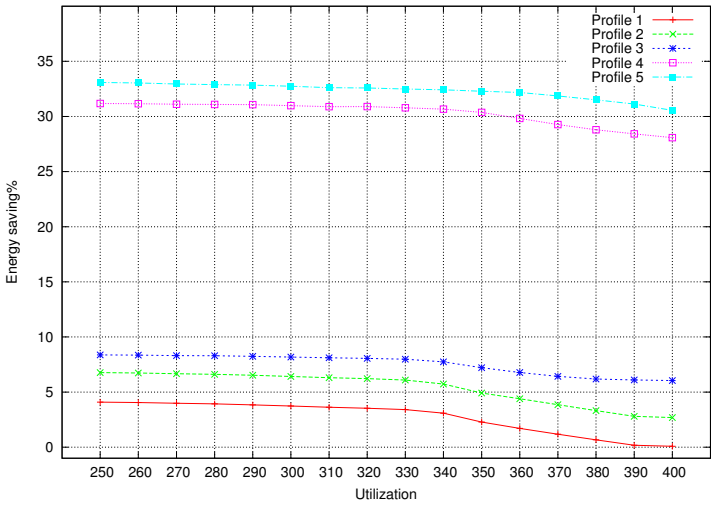


Figure 4.14: Energy saving for MCS profiles

at the maximum frequency. If we consider that profile 0 supposes 0% of performance loss (Fig 4.15):

- Profile 1 increases the performance in relation to profile 0 (the system is 15% closer to be completely filled).
- Profile 2 decreases the performance when DLO tasks are trimmed (performance loss of 10% with respect to profile 0)
- Profile 3 also decreases the performance when RLO tasks are trimmed (performance loss of 30% with respect to profile 0)
- Profile 4 also decreases the performance when DLO tasks are dropped (performance loss of 55% with respect to profile 0)
- Profile 5 increases the performance in relation to profile 4 (increasing computation time of HI tasks by reducing the system frequency) but, with respect to the original profile, there is a performance loss of 40-55%.

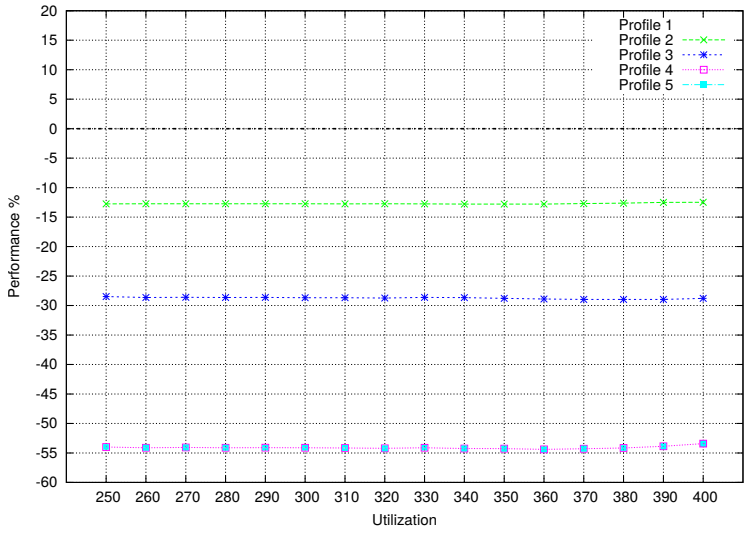


Figure 4.15: Performance loss for MCS profiles

- Number of mappings. Parameter k is calculated as in non MCS. As it is seen in Fig 4.16, obviously the number of mappings increases with profile. It is obvious that the more operations (k) to partitions are needed, the more attempts to fill the cores are done.

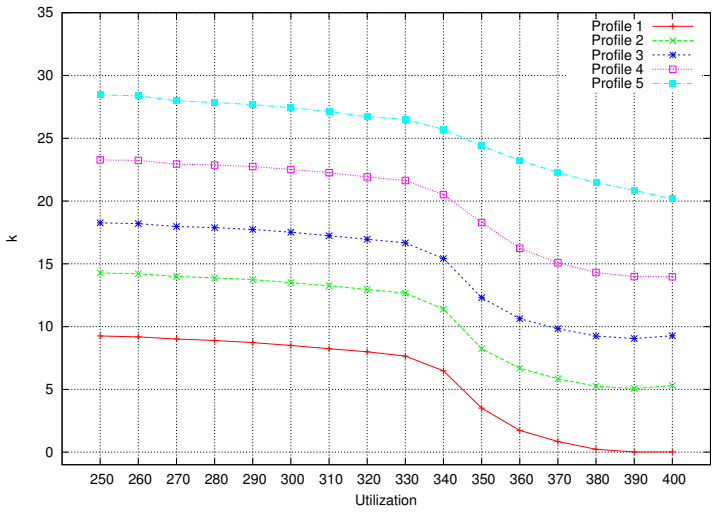


Figure 4.16: k for MCS profiles

Using a CP solver in order to conduct these experiments provides worst solutions than in non MCS. This occurs because the number of constraints increases due to the addition of the criticality of the partition. In most cases, we have spent several days without reaching the solution.

4.5 Practical application: Xoncrete

4.5.1 Introduction

Throughout this chapter, energy saving techniques have been developed for partitioned systems. After experiencing great results, the implementation of previous energy-aware algorithms in a commercial tool has been considered interesting for the development of this thesis.

The motivation on developing a schedulability analysis tool is based on that the system models considered by researchers and those used in real problems not always coincide. There is a wide range of scheduling models (periodic, sporadic, etc.) and analysis (schedulability, sensitivity, resource sharing, etc.). With the purpose of performing the schedulability analysis, different tools exist and are classified depending on their purpose: general or specific.

Xoncrete [**?**] is a tool assigned to meet the ARINC 653 requirements of the aeronautical and aerospatial systems. Thus, it is a specific purpose tool developed to assist the system integrator to build and analyse the schedulability of a partitioned system and to create the cyclic scheduling table, in particular the XtratuM framework.

The purpose of this section is to describe Xoncrete before the implementation of energy saving algorithms and to specify the updates in this commercial tool.

4.5.2 Description of the previous state

Besides the scheduling analysis capabilities, the Xoncrete tool also provides a user friendly interface for capturing and editing all the elements that are part of a partitioned system. It has been specially designed to generate configuration files compatible with XtratuM. The user defines all the elements of the partitioned system using Xoncrete's web interface. Xoncrete maintains its own format to save all the data into a project file. This format is known as *eprj* format file.

The main features of Xoncrete are listed below:

- Powerful configuration and validation tool.
- User friendly interface.
- Partitioned system edition.

- Scheduling analysis capabilities.
- Multi-plan support.
- Advanced health monitoring and error reporting.
- Based on web standards (HTML, CSS, Javascript, AJAX)

The scheduling analysis process covers the following phases (Figure 4.17):

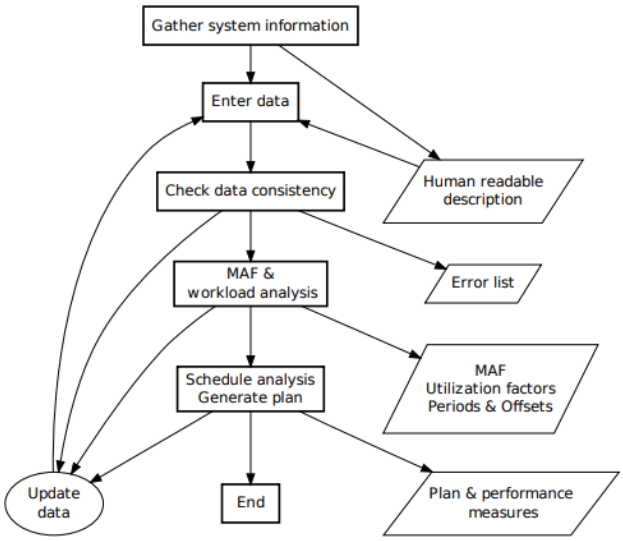


Figure 4.17: Xoncrete work flow.

1. Introduce system parameters and timing requirements.
2. Enter the data to be used in the analysis.
3. Check the consistence of the entered parameters and requirements.

Now, the system can be analysed and the scheduling plan can be generated. The following steps are:

1. Temporal behaviour. Tune the End-to-End flow (ETEF) parameters. An ETEF describes sequence of tasks with temporal attributes. It is the element that defines the periodic behaviour and temporal restrictions of the tasks. This concept will be explained later.

2. Schedule generation. In this phase, the scheduling analysis algorithm is executed and the plan is generated.

Once the plan is generated, all the configuration information is exported as a *xml* file. Xoncrete works as an assistant to generate the configuration file of Xtratum, ready to be used without modifications. The exported configuration file includes all the necessary scheduling information generated by the schedulability analysis module. This file also contains information about the memory layout, communication channels, devices, etc.

The following is the description of modifications done to Xoncrete in order to provide energy aware scheduling to a partitioned system.

4.5.3 Modifications in the tool

One of the results of Section 4.3.2 is the generation of different scheduling plans with different energy consumption and performance loss for a MCS. In this situation, each partition can be executed with a processor frequency and some of them may be removed, depending on their nature. Concepts as criticality of a task, system frequency, energy savings and performance loss are implemented in Xoncrete in order to provide enhanced services to the user.

In order to adequate Xoncrete to energy-aware purposes, some novelties are added to the tool:

- The range of operating frequencies of the cores. This option is found in Edit→System Resources→Hardware Resources (Figure 4.18).

  Different cores can, or can not, run at the same frequency, depending on the system needs. In the SAFEPOWER project scenario, all cores run at the same frequency.

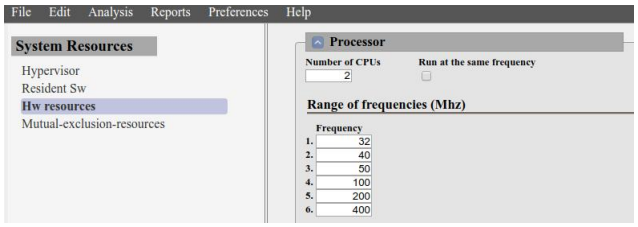


Figure 4.18: System frequencies.

- The range of WCETs per task, depending on the corresponding frequency. This option is found in Edit→Partitions→Name of the partition→Internal tasks and scheduling (Figure 4.19).

- The criticality level of each partition. This option is found in Edit→Partitions→Name of the partition→Internal tasks and scheduling (Figure 4.19).

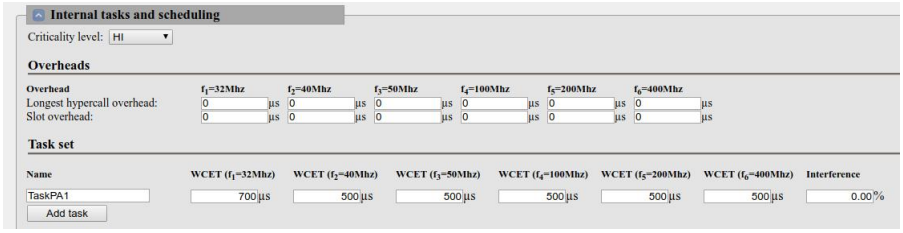


Figure 4.19: Internal tasks and scheduling section

- Selection of the profile for each plan (See Section 4.3.2). They can be selected in Analysis→Temporal analysis(Figure 4.20).

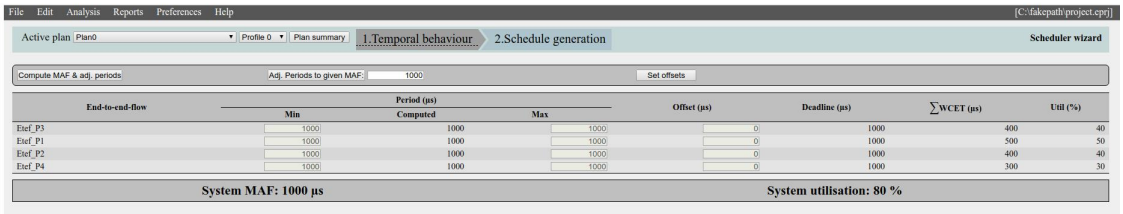


Figure 4.20: Temporal behaviour

Now, an use case is presented in order to show the usage of the tool.

4.5.4 Use case

The main window of Xoncrete tool is shown in Figure 4.21. First, this window is empty until the system designer configures all the system information. The reader should bear in mind that only the aspects related to this thesis are going to be explained in this section.

From now on, the generation of different profiles for a MCS through Xoncrete tool are being explained.

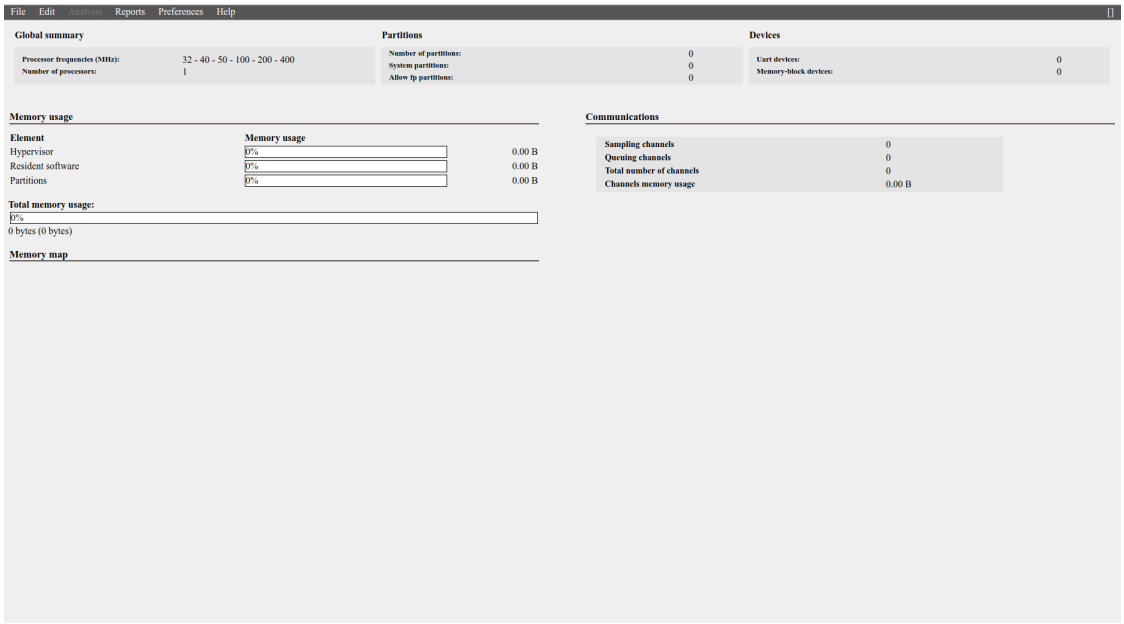


Figure 4.21: Main window of Xoncrete application.

In the first step, the system designer configures the resources: hypervisor, hardware resources as number of processors, range of frequencies per processor, etc. (Figure 4.22) and mutual-exclusion resources.

As this work is framed within SAFEPOWER project [**?**], processor frequencies values correspond to the processor used in the project, i.e., an ARM Cortex-A9 dual core.

Regarding to mutual-exclusion resources, the Stack Resource Protocol [**?**] is used to avoid unbounded priority inversion between tasks.

After describing system resources, partitions are defined (Figure 4.23). Among others, communication ports, internal tasks, interrupts, etc. are set in the partition definition. Tasks are defined by their criticality level, computation times per frequency and effective virtual CPU.

Instead of defining a period per task, Xoncrete introduces the concept of End-to-End flows (ETEFs). An ETEF is defined as a sequence of tasks with temporal attributes and includes the periodic behaviour of the system. It is characterized by a set of tasks, deadline, period and offset. Tasks in an ETEF can belong to different partitions and each task can appear in more than one ETEF. For this reason, the period is adjusted in the ETEF and not in each task. A diagram of an ETEF is

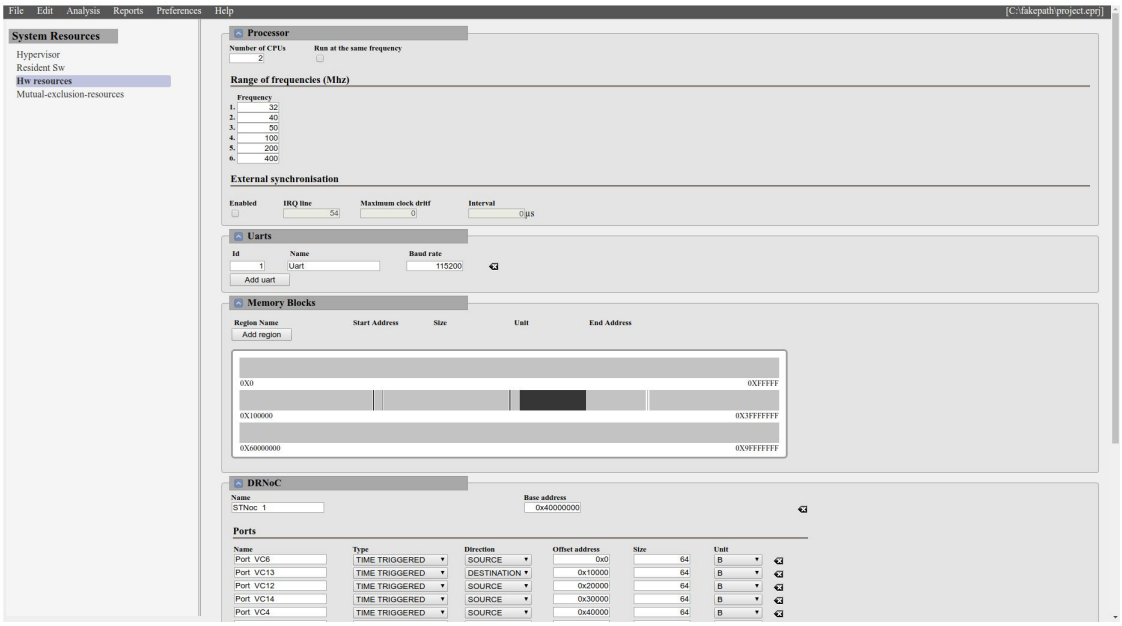


Figure 4.22: Hardware resources edition.

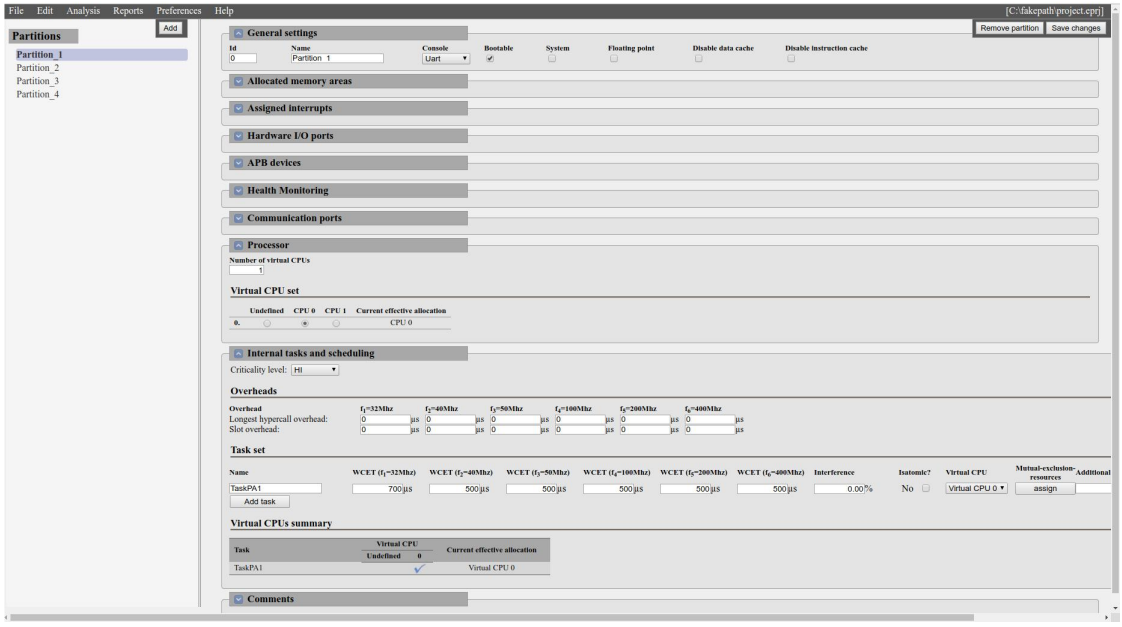


Figure 4.23: Partition edition.

shown in Figure 4.24. If an ETEF is defined by only one task, the ETEF will be equivalent to the definition of a classical single task.

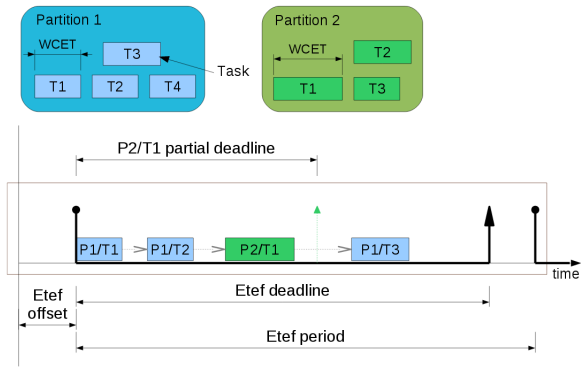


Figure 4.24: End-to-end flow definition.

Instead of defining the period of an ETEF by a single value, Xoncrete allows the user to define it as a range of periods (Figure 4.25) and the tool will be the responsible of the period selection. This selection is made through a novel algorithm [**?**] with the purpose of reducing the major active frame (MAF) of the system, i.e., the least common multiple between the periods of all ETEFs.

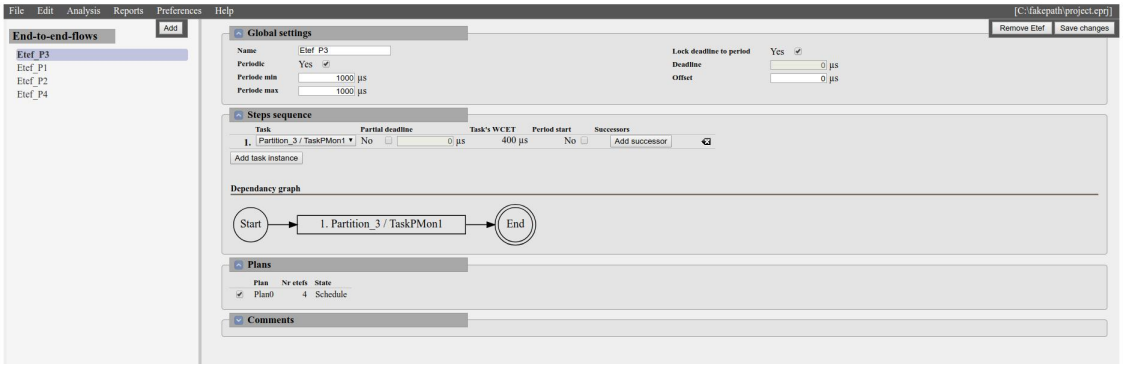


Figure 4.25: End-to-end flow edition.

Once the system parameters are introduced, they are validated to check the data consistency. Then, Xoncrete selects the period for each ETEF among the range of possible values, in such a way as the MAF is reduced in order to be practical. In Figure 4.26, “Compute MAF & adj. periods” will execute this option.

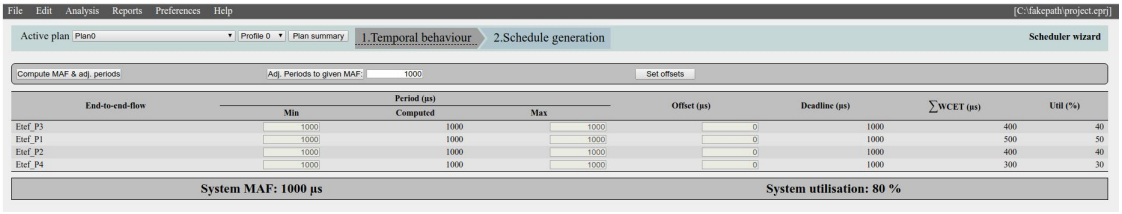


Figure 4.26: Temporal behaviour.

After this calculation, the scheduling plan for each profile can be generated. Let us use as a example a system with 2 cores that allocate 4 partitions in them. Each partition is composed by a task and each task belongs to a different ETEF. The tasks computation time and periods of the ETEFs are described in the Table 4.7:

Table 4.7: Utilizations - Example

	L	$WCET_i^{f1}(\mu s)$	$WCET_i^{f2}(\mu s)$
$T_1(P_1)$	HI	700	500
$T_2(P_2)$	HI	560	400
$T_3(P_3)$	RLO	560	400
$T_4(P_4)$	DLO	420	300

The periods of the ETEFs are set to 1000 μs, instead of giving a range of values per ETEF. The reason is to ensure specific values and to make a comparison between the results obtained in simulation and the results proposed by this tool.

The last step is to generate the plan. The tool provides a graphical representation of it and also statistic information about the schedulability, utilization, partition allocation to cores, effective CPU usage, energy saving and performance loss.

The plan is generated using well known scheduling techniques:

- EDF as the base scheduling policy.
- Minor modifications to the EDF criteria in order to reduce the number of partition context switches.
- SRP to access mutual exclusion resources.

For each profile described in Section 4.3.2, a different scheduling plan, with its corresponding energy savings and performance loss, is generated.

The profile 0 corresponds to the plan without energy management. This means that the system runs at its maximum frequency. As seen in Figure 4.27, partitions 1 and 4 are allocated to CPU 0 and 2 and 3, to CPU 1.

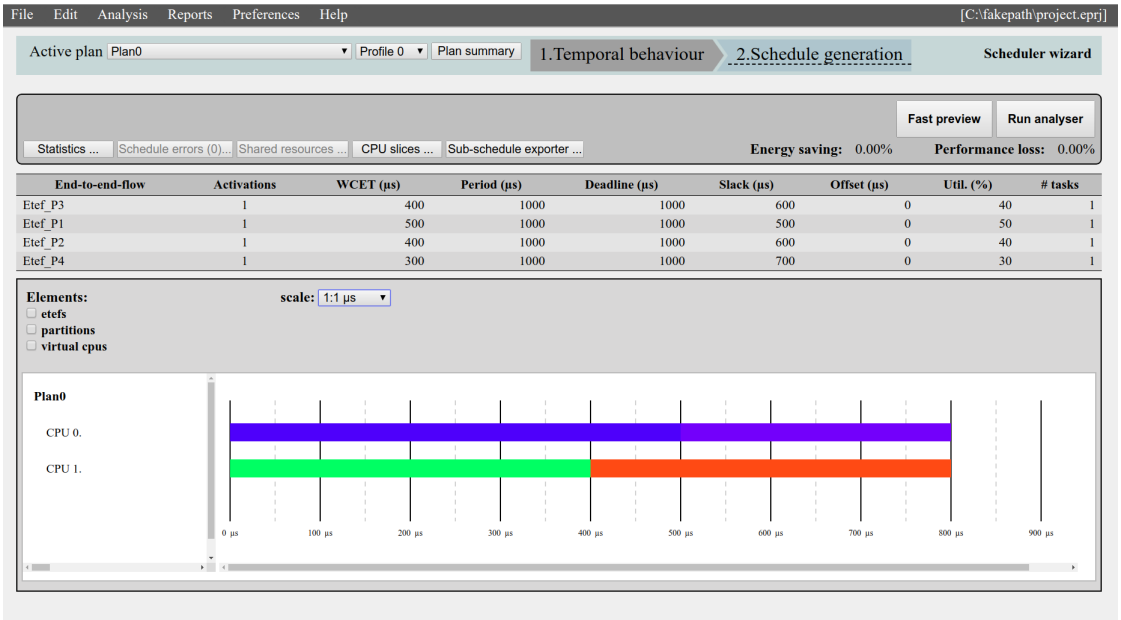


Figure 4.27: Results profile 0.

Profile 1 (Figure 4.28) corresponds to the plan with energy management, in which HI criticality partitions reduce their frequency in order to reduce the total energy consumption. Partitions are selected in decreasing order of utilization, being the ones with higher utilization the ones whose frequency is reduced. This process is repeated until cores are full or there is no other HI partition whose frequency can be reduced. In this example, partitions 1 and 2 reduce their frequency using the EEA algorithm.

Profile 2 (Figure 4.29) executes Trimming operation to DLO partitions, i. e., partition 4 is executed at its minimum frequency but with the computation time of its maximum frequency. It also applies EEA to HI partitions (i.e. Profile 1 + DLO trimming).

Profile 3 (Figure 4.30) executes Trimming operation to RLO , i. e., partition 3 is executed at its minimum frequency but with the computation time of its maximum frequency. It also applies EEA to HI partitions and trims DLO partitions (i.e. Profile 2 + DLO trimming).

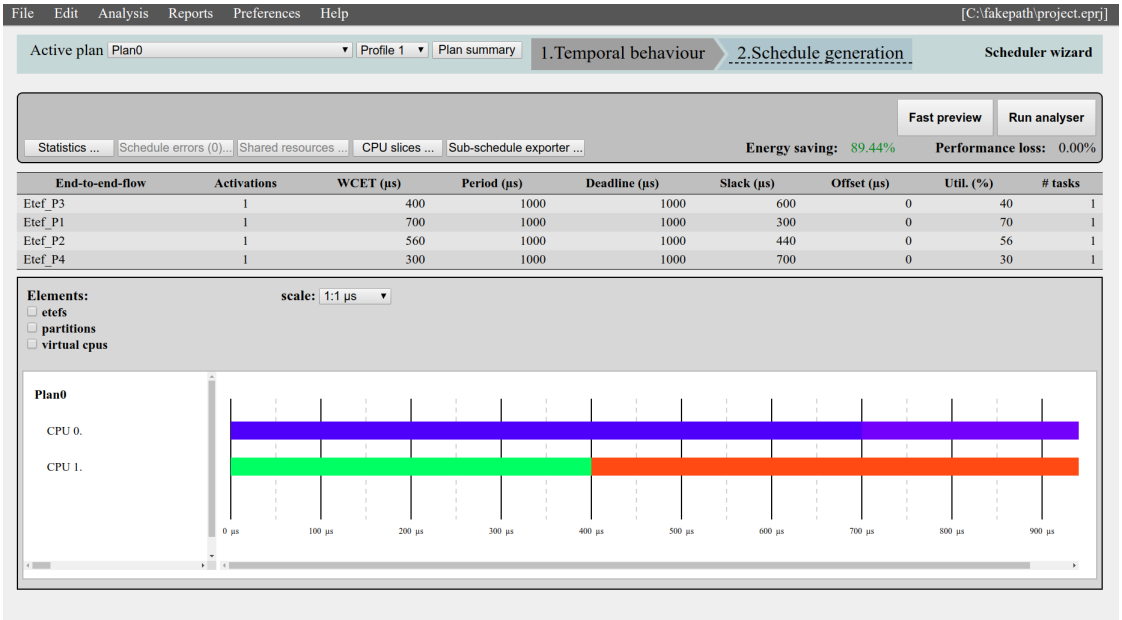


Figure 4.28: Results profile 1.

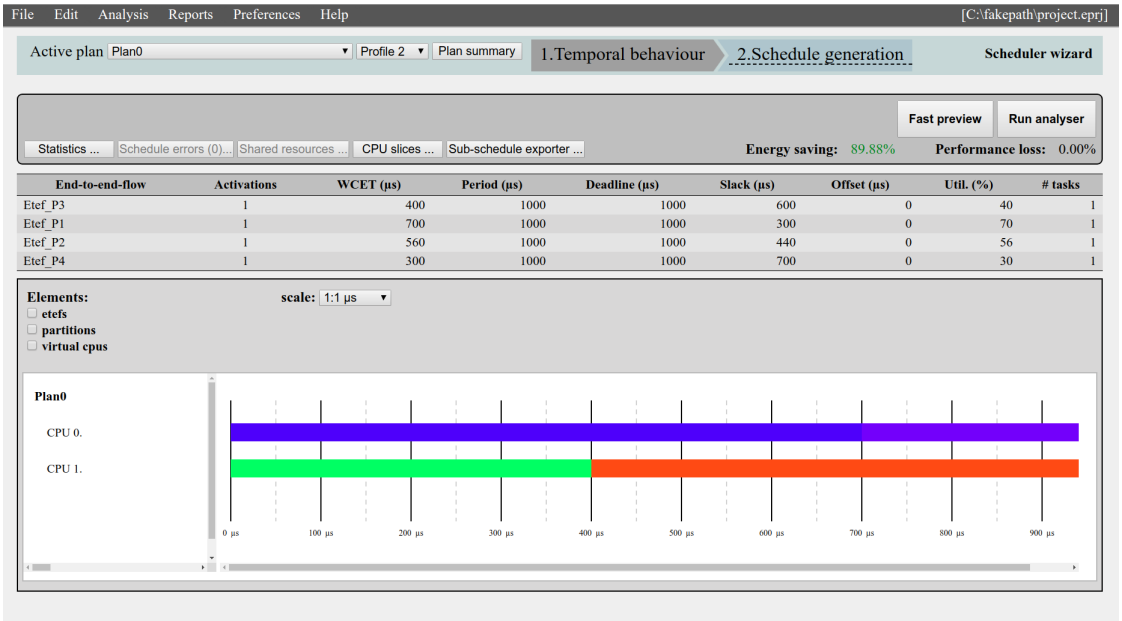


Figure 4.29: Results profile 2.

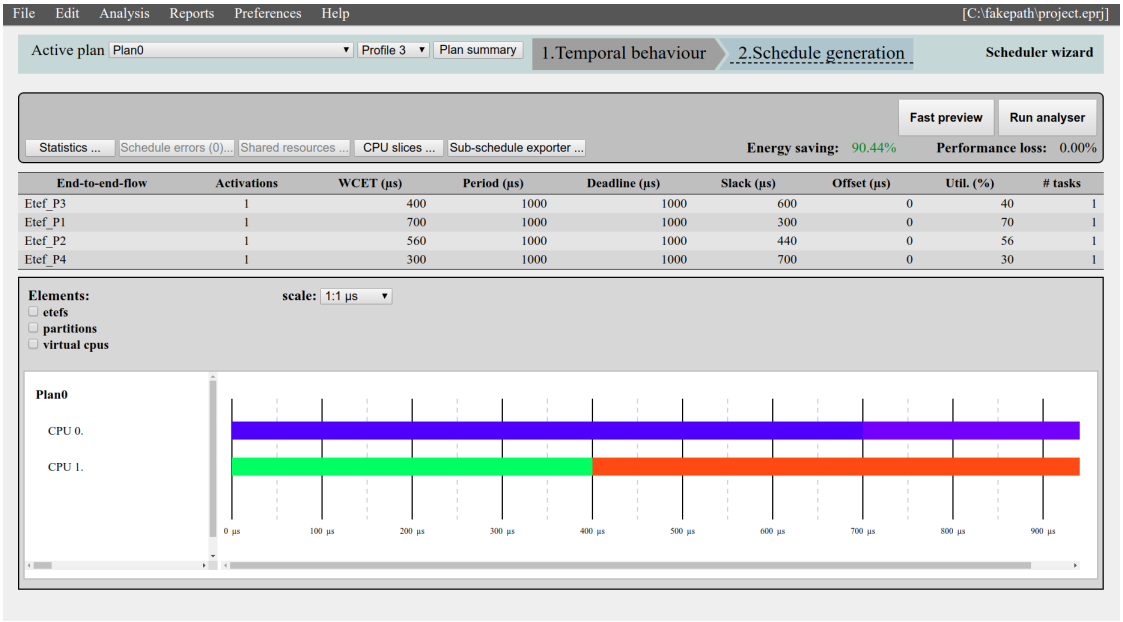


Figure 4.30: Results profile 3.

Profile 4 (Figure 4.31) is obtained dropping DLO partitions and applying EEA to HI partitions (i.e. Profile 1 + DLO dropping).

Finally, profile 5 (Figure 4.32) is obtained dropping DLO partitions, applying EEA to HI partitions and trimming RLO partitions (i.e. Profile 3 + DLO dropping).

The more strategies are applied, the more energy savings are obtained. However, trimming and dropping strategies introduce performance losses in the system. It is clear that, as long as the number of profile increases, the saving and the performance loss also increases. By means of the button “Plan summary”, characteristics of different profiles are shown. It contains information about temporal behaviour, CPU usage, deadline misses, energy saving and performance loss. As seen in Figure 4.33, the percentage of energy savings increases with the profile, except in profile 4, in which only EEA algorithm and dropping strategies have been applied.

As shown in Figure 4.33, Profile 0 is the original plan, without applying energy saving techniques (0.00%) and Profile 5 is the profile with the most energy savings (58.69%). There is not too much difference between profiles 3 and 5 in terms of energy consumption but the difference of performance loss is considerable.

When everything is generated, the main window of Xoncrete changes and now includes the most relevant information about the system model.

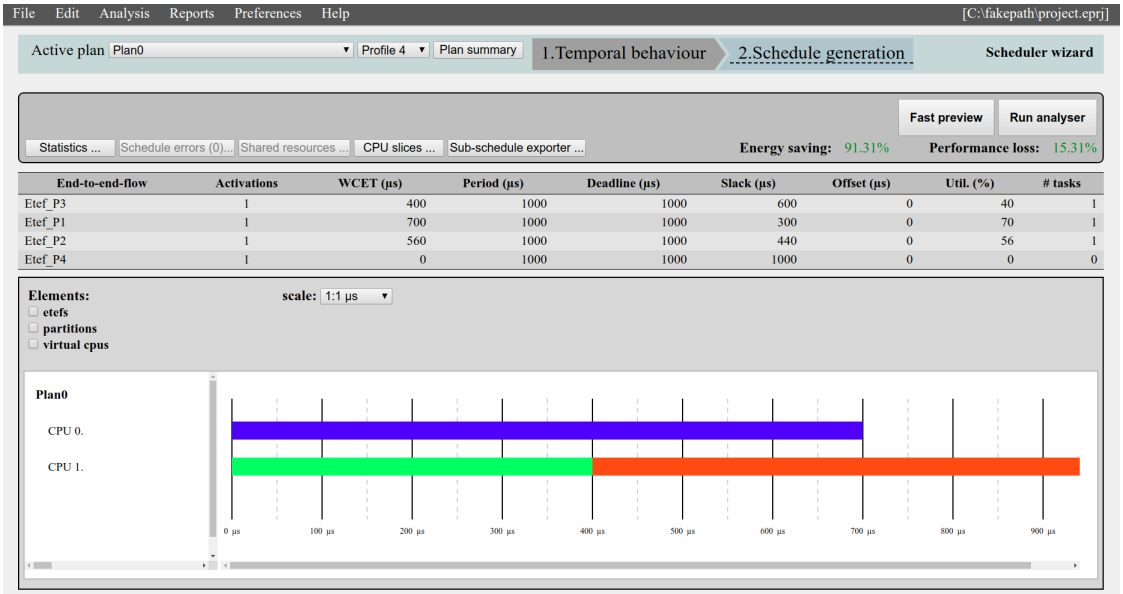


Figure 4.31: Results profile 4.

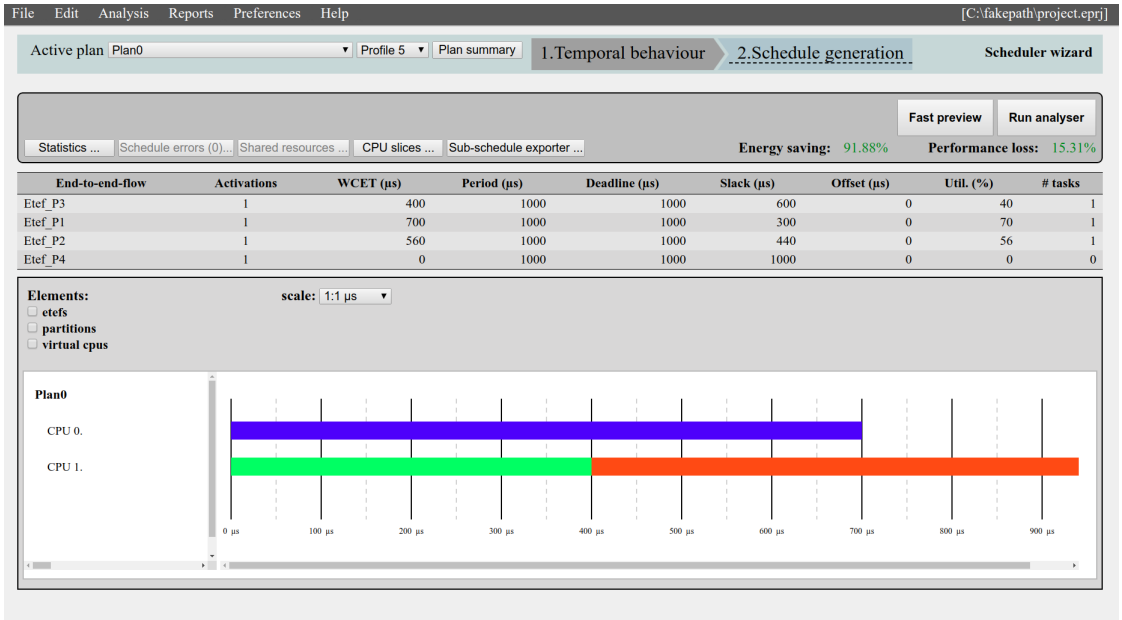
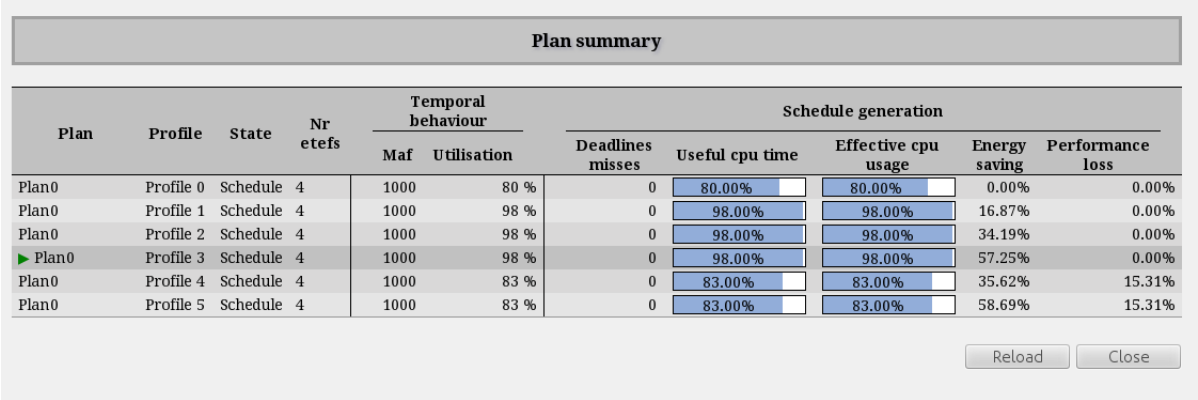


Figure 4.32: Results profile 5.



Plan summary

Plan	Profile	State	Nr etefs	Temporal behaviour		Schedule generation				
				Maf	Utilisation	Deadlines misses	Useful cpu time	Effective cpu usage	Energy saving	Performance loss
Plan0	Profile 0	Schedule	4	1000	80 %	0	80.00%	80.00%	0.00%	0.00%
Plan0	Profile 1	Schedule	4	1000	98 %	0	98.00%	98.00%	16.87%	0.00%
Plan0	Profile 2	Schedule	4	1000	98 %	0	98.00%	98.00%	34.19%	0.00%
▶ Plan0	Profile 3	Schedule	4	1000	98 %	0	98.00%	98.00%	57.25%	0.00%
Plan0	Profile 4	Schedule	4	1000	83 %	0	83.00%	83.00%	35.62%	15.31%
Plan0	Profile 5	Schedule	4	1000	83 %	0	83.00%	83.00%	58.69%	15.31%

Reload Close

Figure 4.33: Plan summary.

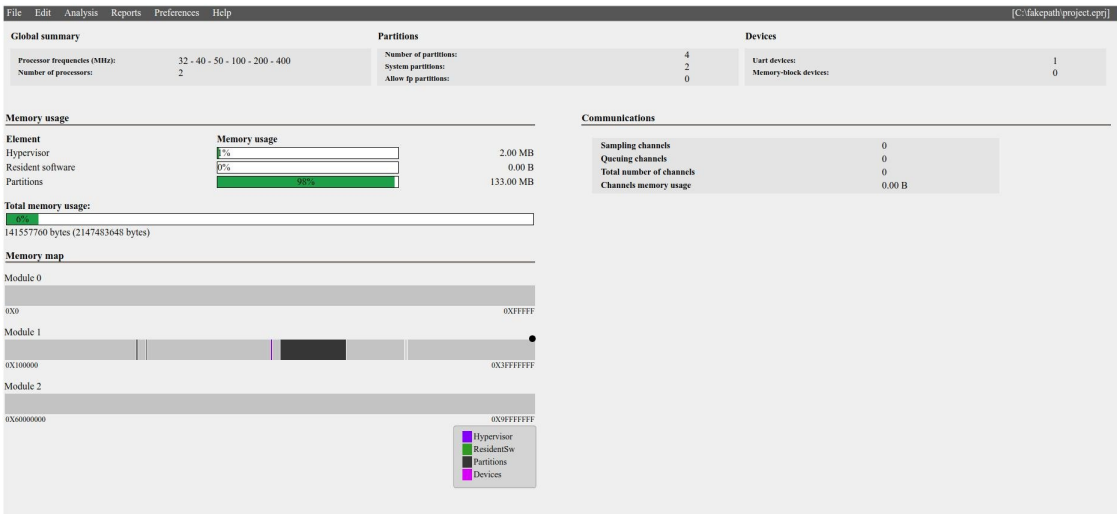


Figure 4.34: Main window of Xoncrete application. Updated model.

As stated at the beginning, Xoncrete's last step consists of exporting all the necessary scheduling information to Xtratum configuration file.

The xml file that corresponds to the example mentioned in this chapter is shown in Figure 4.35.

```
- <SystemDescription version="1.0.0" name="no_name">
  + <HwDescription></HwDescription>
  + <XMHypervisor console="Uart"></XMHypervisor>
  + <PartitionTable></PartitionTable>
  + <Channels></Channels>
  </SystemDescription>
```

Figure 4.35: Reduced information in xml file.

As seen in Figure 4.35, this file contains hardware description, xm hypervisor information, partition table and communication channels information. The most relevant part for this work is detailed in "HwDescription".

As seen in Figure 4.36, the temporal description in the form of a cyclic scheduling plan is included for each core. Each plan is composed by a set of slots (pieces of time) characterised by an identifier, start time, duration, partition executed during this time and the system frequency in this slot.

This file is provided as an input to the hypervisor and is ready to be used without modifications. The goal is to generate a short cyclic plan which can be used as an ARINC 653 partition plan.

```
- <SystemDescription version="1.0.0" name="no_name">
  - <HwDescription>
    + <MemoryLayout></MemoryLayout>
    - <ProcessorTable>
      - <Processor id="0" frequency="400Mhz">
        - <CyclicPlanTable>
          - <Plan name="Plan0-0" id="0" majorFrame="1000us">
            + <!---->
              <Slot id="0" start="0us" duration="500us" partitionId="0" frequency="400Mhz" vCpuId="0"/>
            + <!---->
              <Slot id="1" start="500us" duration="300us" partitionId="3" frequency="400Mhz" vCpuId="0"/>
            </Plan>
          - <Plan name="Plan0-1" id="1" majorFrame="1000us">
            + <!---->
              <Slot id="0" start="0us" duration="700us" partitionId="0" frequency="200Mhz" vCpuId="0"/>
            + <!---->
              <Slot id="1" start="700us" duration="300us" partitionId="3" frequency="400Mhz" vCpuId="0"/>
            </Plan>
          - <Plan name="Plan0-2" id="2" majorFrame="1000us">
            + <!---->
              <Slot id="0" start="0us" duration="700us" partitionId="0" frequency="200Mhz" vCpuId="0"/>
            + <!---->
              <Slot id="1" start="700us" duration="300us" partitionId="3" frequency="32Mhz" vCpuId="0"/>
            </Plan>
          - <Plan name="Plan0-3" id="3" majorFrame="1000us">
            + <!---->
              <Slot id="0" start="0us" duration="700us" partitionId="0" frequency="32Mhz" vCpuId="0"/>
            + <!---->
              <Slot id="1" start="700us" duration="300us" partitionId="3" frequency="32Mhz" vCpuId="0"/>
            </Plan>
          - <Plan name="Plan0-4" id="4" majorFrame="1000us">
            + <!---->
              <Slot id="0" start="0us" duration="700us" partitionId="0" frequency="200Mhz" vCpuId="0"/>
            </Plan>
          - <Plan name="Plan0-5" id="5" majorFrame="1000us">
            + <!---->
              <Slot id="0" start="0us" duration="700us" partitionId="0" frequency="32Mhz" vCpuId="0"/>
            </Plan>
          </CyclicPlanTable>
        </Processor>
      + <Processor id="1" frequency="400Mhz"></Processor>
      </ProcessorTable>
    + <Devices></Devices>
    </HwDescription>
```

Figure 4.36: Temporal description in xml file.

4.6 Theoretical energy characterization

As energy management is a very active research area in the recent years, many approaches to minimize energy consumption under the constraint that all tasks meet their deadlines have been proposed. For this reason, this whole chapter has focused on presenting energy saving techniques in partitioned multicore systems. First, heuristic algorithms were proposed and a simulator that consolidates previous results was implemented.

After evaluating the good results proposed by this heuristic, a theoretical basis is considered essential for their validation. The following aims to cope with new challenges when applying two widely used techniques for reducing energy consumption: Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management.

Thanks to techniques as DVFS, the control of the frequency in a partitioned system is a very important issue in order to reduce the energy consumption of the global system. Reducing the system frequency supposes a decrease in the energy due to dynamic power but the time the processor is busy increases. Then, which is the best relation between frequency and the time the processor works in terms of energy consumption? All existing works assume that frequency and temporal load are linearly related. Is this the model that best fits these two variables?

Thus, the following section attempts to answer the previous questions, being the main contributions of this section:

1. Almost all works in literature assume that the relation between temporal cost and the inverse of the frequency is linear. Our objective is to propose a non-linear relation between these parameters and deduce which of both theories is more suitable in terms of energy consumption.

2. To obtain a model that relates computation time (utilization), frequency and energy consumption in a partitioned system.

3. Demonstrate that there are different points with different utilization and frequency that provide the same energy consumption. This is very useful for system designers, to be able to choose which frequency reduces the energy consumption of the system.

First, an analysis of the current frequency-time model is made. For this purpose, some experiments, in order to evaluate how realistic the model in almost all works is, are conducted. After evaluating this model, a different approach that best fit the experiments in our platform are made.

Next, the function that relates the three parameters time, frequency and energy of the system is calculated.

Finally, optimization techniques to this obtained function are applied in order to calculate the optimal points.

4.6.1 Relation between time and frequency

There are a lot of works ([**?**],[**?**]) that assume a linear relation between computation time of a partition (i.e. partition utilization) and the inverse of the frequency. However, it is known that this simplification is not very realistic. We want to analyse how far the linear model is from reality. In order to obtain a more realistic relationship, we have made some real measurements in an ARM Cortex-A9 dual core. The experimental measurements have been made in the context of SAFEPOWER project, whose goal is to enable the development of low power mixed-criticality systems through the provision of a reference architecture, platforms and tools to facilitate the development, testing, and validation of these kinds of systems according to the market needs. In these experiments, intensive simulations have been performed to obtain a large number of computation time measurements. Details about these experiments can be found in [**?**]. The results are as follows:

Table 4.8: Computation time measurements at different frequencies

$Frequency(MHz)$	$Cost(us)$
400	7
200	10
100	17
50	35
40	44
32	56

We suppose that these measurements are periodic in order to translate the computation times to utilization. If we make this conversion and then we represent the values, the results are represented in Figure 4.37. We depict frequency versus the inverse of the time, in order to adapt our study to [**?**],[**?**] and elsewhere, where time intervals are inversely proportional to frequency.

In this section we are going to calculate both, linear and non-linear approximation, and then we will make a comparison between energy costs in both scenarios.

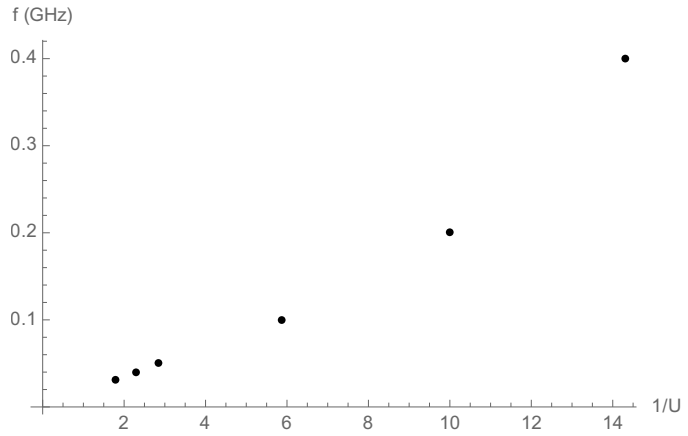


Figure 4.37: Experimental relation between utilization and frequency.

4.6.1.1 Linear approximation

There are a lot of methods to obtain an approximation of a set of points. As time intervals are inversely proportional to the system frequency, this relation is $t = k/f$. Adjusting *k* value, the function that best fits the points in table 4.37 is:

$$f(U) = \frac{0.025}{U} \tag{4.6}$$

4.6.1.2 Non-linear approximation

Due to the shape of the function in Figure 4.37, it is obvious that vertical and horizontal asymptotes exist. It is not really accurate to follow this behaviour through a line, but rather with a rational function. In this section, we are going to calculate the best rational function that fits the points in Figure 4.37.

In mathematics, Thiele's interpolation formula [**?**] is the way to define a rational function $f(x)$ from a finite set of inputs x_i and their function values $f(x_i)$. (The step-by-step solution is found in Appendix A.1. Thus, the obtained function is:

$$f(U) = \frac{0.016U + 0.02688}{U} \tag{4.7}$$

and, from now, $a = 0.016$, $b = 0.02688$, $c = 1$ and $d = 0$ (a, b, c and d are parameters of a general rational function). If we represent these two functions in the same graph[1] (Figure 4.38), it is easy to see that this non-linear function (in red)

[1] In order to represent frequency versus the inverse of the utilization (as other works), we convert

is the one that best fits to the experimental results.

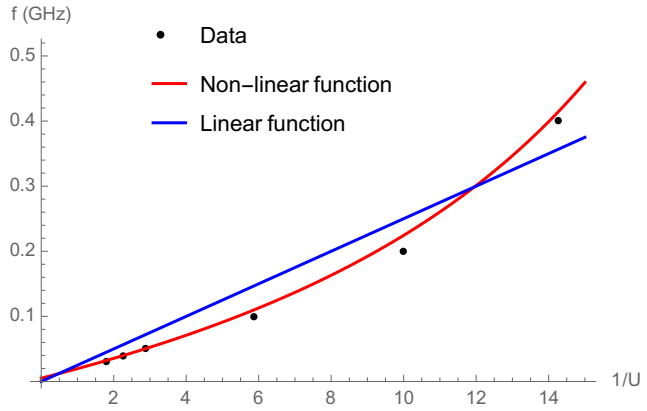


Figure 4.38: interpolation functions between utilization and frequency.

In spite that the non-linear function provides better results, to calculate and implement the linear function is usually more efficient, as regards the time taken in calculating the parameters. However, the non-linear model is adjusted to best correspond to the practical results.

Now, the energy of the system is introduced and the model that relates these three parameters is deduced.

4.6.2 Energy-Time-Frequency relation

In order to calculate the optimal energy consumption in a theoretical way, first a model that relates the three involved variables, computation time, system frequency and energy consumption, is deduced. We assume the power model defined in Section 4.2.3 as a starting point.

Definition 4.6.1. As power is how quickly energy is used or transferred, the energy function can be expressed as:

$$E(t) = P(t) \cdot t \tag{4.8}$$

Then, using the equation 4.1, let us define the energy function:

$$E(f,t) = (P_s + \beta f^{\alpha})\, t \tag{4.9}$$

equations 4.6 and 4.7 into their inverses.

The time *t* the processor is busy can be calculated as $MAF \cdot U$. Then, applying this definition to equation 4.9, the energy-time-frequency function is defined as:

$$E(U, f) = (P_s + \beta f^{\alpha})\, MAF \cdot U \tag{4.10}$$

Section 4.6.3 presents an analysis of the Equation 4.10, in order to calculate the optimal energy consumption. That equation relates how the utilization and the frequency affect to the energy consumption of the system. Different combinations of points (U,f) may provide same values of energy, as will be demonstrated later.

However, there is a critical frequency below which it is not beneficial to reduce frequency energy-wise [**?**]:

$$f_{crit} = \sqrt[\alpha]{\frac{P_s}{\beta\,(\alpha - 1)}} \tag{4.11}$$

The critical frequency represents the frequency that minimizes the energy consumption for execution when the overhead for sleeping is considered negligible, shown in [**?**].

With the functions presented in these sections, let us find the optimal solution of the energy consumption through mathematical analysis and compare the results between linear and non-linear interpolation.

4.6.3 Optimal energy consumption through mathematical analysis

In general, this problem consists of finding the relative extrema (maximum and minimum) of a multivariable function $f(x, y)$ subject to a constraint $g(x, y) = 0$ and defined in a specific area *S*. In this kind of situations, the solution is found following two steps:

- Search relative points of $f(x, y)$ and select only that ones that belong to *S*.
- Use the constraint $g(x, y) = 0$ applying the method of Lagrange multipliers.

In an optimization problem, there are several methods to find relative extrema. We are going to use the method of Lagrange multipliers. It involves much greater difficulty than others, but we avoid the problem of symmetry losses offered by the method of variable substitution. This method is used in multivariable functions subject to equality constraints. Defining the problem as a two choice variables function $f(x, y)$ subject to $g(x, y) = 0$, the Lagrange function is defined as:

$$\mathcal{L}(x, y, \lambda) = f(x, y) - \lambda \cdot g(x, y) \tag{4.12}$$

, being λ a Lagrange multiplier. This method says that those points where the original function has relative extrema conditioned to some constraints, are placed between the stationary points of the consequent Lagrange function.

Once this has been presented, we are going to define our problem. Let us use the equation 4.12, being $f(x, y) = E(U, f)$ the energy function and $g(x, y) = 0$ the constraint, i.e. the relation between *f* and *U* (see Section 4.6.1).

Let us look at the problem from the geometric point of view. Graphically, $E(U, f)$ function, without any constraints is shown in Figure 4.39.

To limit the problem, we are going to study the two possible relations between f and U mentioned before. At this point we deal with each relation separately and in a specific region.

4.6.3.1 Linear relation.

In this section, the constraint $g(x, y) = 0$ mentioned before is the explicit equation $f - k/U = 0$.

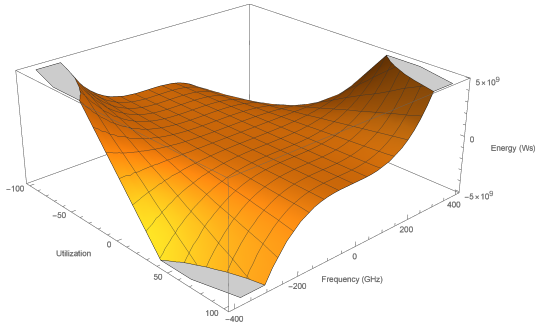


Figure 4.39: Graphical representation of the general function E(U,f) without constraints.

Definition 4.6.1. The problem we have to solve is:

$$\begin{cases} E(U,f) = (P_s + \beta \cdot f^{\alpha}) \cdot MAF \cdot U \quad \text{subject to} \\ f - \frac{k}{U} = 0 \end{cases} \tag{4.13}$$

in the region

$$S = \left\{(U,f) \in \mathbb{R}^2 \mid 0 \leq U \leq 1, f_{crit} \leq f \leq f_{max}\right\} \tag{4.14}$$

The region *S* in defined as follows:

- The utilization is a value between 0 and 100% of the maximum capacity.
- The system frequency is a value between the critical frequency defined in equation 4.11 and the maximum frequency in a processor. For the processor used, we assume a maximum frequency of $f_{max} = 2GHz$.

The steps to find the relative extrema in a constrained problem are applied also in the situation of non-linear approximation. For this reason, we are going to develop it in this section and it will be replicated in the next section.

Firstly, we are going to find the relative points of $E(U,f)$ solving the system $\nabla E = (0,0)$. This is done setting each partial derivative equal to 0. Then, we will select those points that belong to the region S.

$$\frac{\partial E}{\partial U} = 0 \qquad \rightarrow \qquad (P_s + \beta \cdot f^{\alpha})\, MAF = 0$$
$$\frac{\partial E}{\partial f} = 0 \qquad \rightarrow \qquad MAF \cdot \beta \cdot \alpha \cdot U \cdot f^{\alpha-1} = 0$$

These two equations compose a linear system of equations but the solution $\notin S$. So, we discard this point.

Secondly, we are going to use the constraint to find other relative extrema. In this work, the Lagrange function defined in Equation 4.12 is:

$$\mathcal{L}(U, f, \lambda) = E(U, f) - \lambda \cdot (f - \frac{k}{U}) \tag{4.15}$$

The next step consists on setting the gradient $\nabla\mathcal{L}$ equal to the 0 vector.

$$\begin{aligned} \frac{\partial\mathcal{L}}{\partial U} = 0 \quad &\rightarrow \quad (P_s + \beta \cdot f^{\alpha}) \cdot MAF - \frac{\lambda \cdot k}{U^2} = 0 \\ \frac{\partial\mathcal{L}}{\partial f} = 0 \quad &\rightarrow \quad \beta \cdot \alpha \cdot MAF \cdot U \cdot f^{\alpha-1} - \lambda = 0 \\ f - \frac{k}{U} = 0 \quad &\rightarrow \quad f - \frac{k}{U} = 0 \end{aligned} \tag{4.16}$$

This three equations compose a linear system of equations that has to be solved. Setting the values $MAF = 2000ms$, $P_s = 0.8$W, $\alpha = 3$ and $\beta = 1\frac{W}{GHz^{\alpha}}$ [**?**], the relative extremum (U_0, f_0) is:

$$\begin{aligned} U_0 &= \frac{1}{8\sqrt[3]{50}} \\ f_0 &= \sqrt[3]{\frac{2}{5}} \end{aligned} \tag{4.17}$$

The relative extremum (U_0, f_0) is a minimum of E function. Please note that because of space limitations, missing content is located in Appendix A.2.

Graphically, it is not possible to draw the linear constraint $f - \frac{k}{U} = 0$ with the general values of *k*. For this reason, we are going to apply the previous values in Equation 4.6 with $k = 0.025$.

With this *k* value, the graph of the constraint is depicted in Figure 4.40.

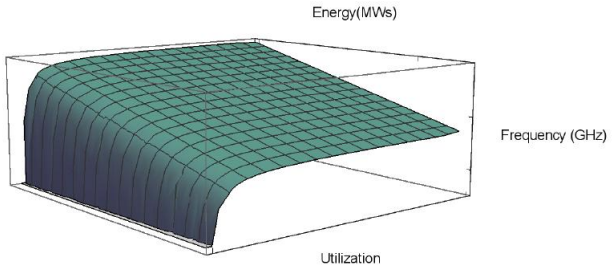
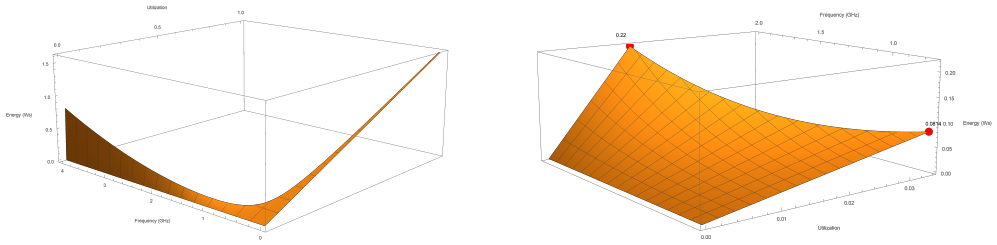


Figure 4.40: Linear constraint in region S.

Drawing the function and the constraint in the same graph (Figure 4.41a), it is seen the intersection. Depicting both in section S, the relative extrema are placed (Figure 4.41b).



(a) Graphical representation of the function E(U,f) and the constraints

(b) Both in region S

Figure 4.41: Relative extrema of the function E(U,f) with different m and n.

From Figure 4.41b and remembering that $E \propto U \cdot f^{\alpha}$, it is easy to see that energy increases in two ways:

- Increasing utilization. It means that frequency is hardly reduced to accomplish equation 4.7.
- Increasing frequency. It means that utilization is reduced.

In both situations, increasing one of the parameters means decreasing the other one, i.e., there is no point in graph where both frequency and utilization are big values. Obviously, if utilization or frequency is zero, energy consumption is also zero.

Moreover, there are different pairs of points that correspond to the same energy level. These points will be in the same plane of energy.

In the experimental scenario, substituting k in Equation 4.17, the solution is shown in Table 4.9.

This solution ensures that the minimum energy consumption occurs when the system runs at its critical frequency (as all theories ensure, for example [**?**]). Maximum energy consumption occurs at maximum frequency and these values coincide with those calculated theoretically in Equation 4.17.

Table 4.9: Maximum and minimum energy consumption in linear approximation ($k = 0.025$).

Frequency (GHz)	Utilization	Energy (Ws)	Point
0.7375	3.39%	0.0814	MIN
2	1.25%	0.22	MAX

In this situation, the relative extremum (U_0, f_0) provided in Equation 4.17 solves the problem directly. And these points coincide with the ones depicted in Figure 4.41b.

4.6.3.2 **Non-linear relation.**

In this section, the constraint $g(x, y) = 0$ mentioned before is the explicit equation of a rational function $f - \frac{aU+b}{cU+d} = 0$.

Definition 4.6.2. The problem we have to solve is:

$$\begin{cases} E(U, f) = (P_s + \beta \cdot f^{\alpha}) \cdot MAF \cdot U \quad \text{subject to} \\ f - \frac{aU+b}{cU+d} = 0 \end{cases} \tag{4.18}$$

in the region $S = \{(U, f) \in \mathbb{R}^2 \mid 0 \leq U \leq 1, f_{crit} \leq f \leq f_{max}\}$.

Following the method of Lagrange multipliers as in previous sections, we finally obtain a fourth degree equation as follows:

$$\begin{aligned} &1.6\,(cU+d)^4 + 2\,(aU+b)^3\,(cU+d) + \\ &6U\,[a\,(cU+d) - c\,(aU+b)]\,(aU+b)^2 = 0 \end{aligned} \tag{4.19}$$

With four parameters and with this degree, solving the equation is a really complicated process. Moreover, graphically, it is not possible to draw the constraint with the general values of *a*, *b*,*c*, and *d*. Let use the example presented in Section 4.6.1 and Equation 4.7 in order to fix the parameters: $a = 0.016$, $b = 0.02688$, $c = 1$ and $d = 0$.

With these values, the graph of the constraint is depicted in Figure 4.42. Equation 4.19 with the previous parameters presents four solutions: two of them are imaginary and the others are:

- $U_0 = 0$. It represents the minimum energy consumption (i.e. 0Ws) but this solution is not interesting in the problem.

- $U_0 = 0.036878$, that belongs to the region S. Then, the critical point is $(U_0, f_0) = (0.036878, 0.745)$, where the energy consumption is 0.0895 Ws.

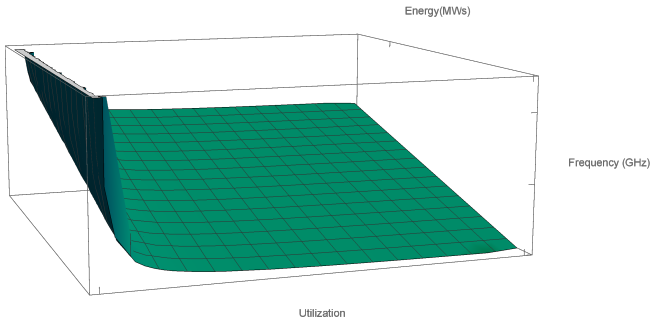


Figure 4.42: Non-linear constraint in region S.

The intersection between the energy function and the constraint in region S is another curve as depicted in 4.43.

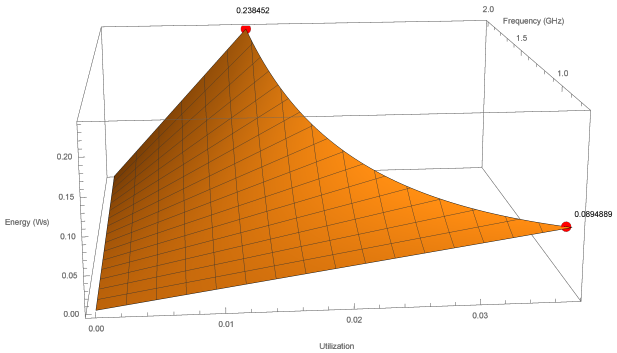


Figure 4.43: Energy function with non-linear constraint in region S.

It is easy to see that there are two relative extrema, both depicted in Figure 4.43. In Table 4.10 results are shown. It is important to observe that the minimum energy consumption point coincides with the calculated mathematically.

Predictably, the maximum energy consumption occurs when processor is running at its maximum frequency and minimum consumption, when it runs at its critical frequency, as was also demonstrated in [**?**].

Table 4.10: Maximum and minimum energy consumption in non-linear approximation.

Frequency (GHz)	Utilization	Energy (Ws)	Point
0.744	3.6%	0.0895	MIN
2	1.35%	0.238	MAX

4.6.3.3 Comparison between linear and non-linear approximation.

All works assume that the relation between cost in time of a partition and system frequency is linear. Previous results show that linear interpolation offers good results in terms of energy consumption. But this approximation is really sensitive to the range of frequencies in which we develop the study. The selection of the parameter k also can variate the solution.

However, with rational approximation this problem does not exist. Even though this approximation requires more calculus knowledges, the obtained function provides a solution that fits better with the problem of energy consumption.

Works consider that maximum energy consumptions takes place at maximum frequency system and, minimum energy consumption at critical frequency of the system. Thanks to rational approximation, these conclusions have been verified.

4.7 Conclusions

In this chapter, the problem of partition allocation in mixed-criticality systems when the goal is to reduce energy consumption has been addressed from two different points of view.

In the first part of the chapter, instead of focusing on new scheduling algorithms to adjust frequency in order to save energy, a partition to CPU allocation that takes into account not only the different frequencies at which the CPU can operate, but the level of criticality of the partitions, is proposed. A different mixed-criticality model is also proposed, instead of the well-known Vestal model. The motivation is to cope with the requirements imposed by the applications used in the avionics and railway sector, since the results of this research will be applied in H2020 project SAFEPOWER with demonstrators in these two sectors.

An allocation method for real-time systems of the same criticality is presented and extended for mixed-criticality systems. First, an Energy Efficient Allocator is defined and then, the extension based on combining two strategies is defined: dropping partitions that are not mandatory in extreme low power situations and reducing the bandwidth of mandatory LO partitions. In the general method, an energy saving up to a 5% is achieved. In the extension to MCS, a 35% saving is achieved at the expense of losing performance of LO partitions. A set of profiles are proposed, so at run time the system has to decide to switch to a more energy conserving profile depending on the power sensor values. In spite of the fact that this is not an exact method, it provides a faster feasible solution, with similar results to the optimal solution.

In the second part of the chapter, a theoretical basis of previous assumptions in order to consolidate them is proposed. In this sense, the relation between computation time and CPU frequency is studied in order to find the best effective approach. To do that, measurements of different functions at different frequencies have been taken on a real platform. This assumption is used to build a mathematical model of energy consumption, frequency and computation time (utilization). The goal is to find the points (frequency, utilization) that minimize energy consumption. The same model is built assuming a linear relationship between frequency and computation time. The comparison between both models confirms that both the linear and non-linear assumptions are realistic. Both provide the same solution: when

the system runs at its maximum frequency, it consumes the maximum energy and when the system runs at its critical frequency, it consumes the minimum energy.

To conclude, energy-aware aspects have been considered in this work, presenting new proposals addressed to mixed criticality systems. Battery-operated devices, supplying computing in space or thermal issues make necessary to address the challenge to minimize energy consumption. This chapter has made a new contribution to this field.

CHAPTER

5

Conclusions

According to the presented work and the detailed experiments and results, the final conclusions can be established. First, the introduced developments are summarized, while a comparison between the initial goals and the obtained achievements characterizes the degree of success of this proposal.

Next, the future work derived from this contribution is presented in order to improve the actual performance and bring new functionalities to the system. Finally, a list of published articles is related to the chapters and sections introduced along this work. Furthermore, the research projects, which frame the development of this thesis, are presented. The layout of the conclusion chapter is depicted in Figure 5.1.

5. CONCLUSIONS

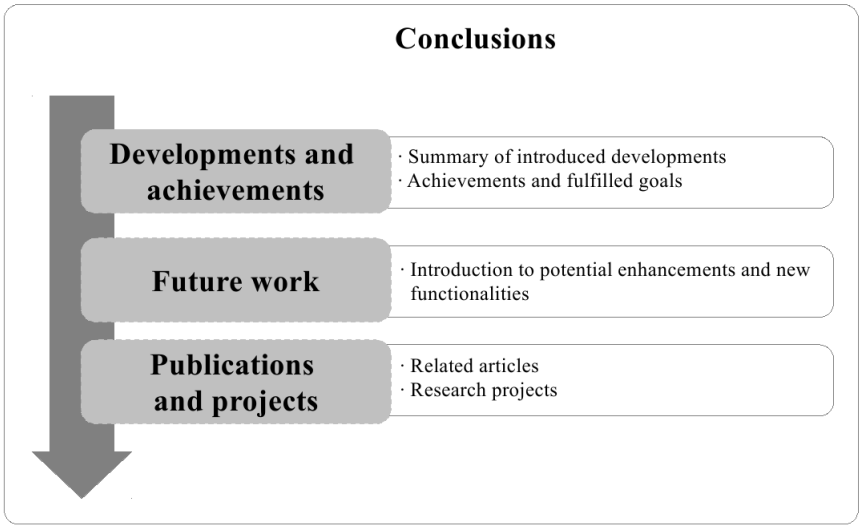


Figure 5.1: Chapter 5 layout

5.1 Developments and Achievements

The main objective of this thesis has been to contribute to the analysis of a partitioned and hierarchical real-time system. In particular, the scheduling of systems in which different criticalities coexist has been addressed.

The main contributions of this thesis are divided into two main groups: on the one hand, the generation of offline scheduling plans for partitioned and hierarchical real-time systems and, on the other hand, energy-aware techniques for multicore real-time systems. Thus, the contributions of this thesis are explained in detail below.

The contributions related to the generation of offline scheduling plans are:

- Study of the schedulability of a two level hierarchical real time system, where local level tasks are scheduled under EDF policy whereas the global level does not follow any known scheduling policy.
- Calculation of CPU supply functions that ensure the schedulability of task sets in this scenario. Specifically, two different functions are defined: firstly, a CPU supply which offers CPU when tasks are released (*gsbf*) and, secondly, a CPU supply which offers CPU as late as possible (*msbf*), meeting the deadlines in both situations and providing the minimum CPU.
- As a consequence of that, the proposed method can be used to check if any set of time intervals provided by the SI can be accepted by the PD or not.
- In spite authors believe that there is no other similar work in literature, this proposal is compared with other existing methods. This is because the proposed work can be applied in the case of any known existing server in the global level.

The contributions related to energy-aware techniques are:

- Study and analysis of classical Vestal model for mixed criticality systems. After exposing Vestal's characteristics that make this model non suitable in our scenario, the model that best fits our project requirements is presented.
- Definition and implementation of an energy efficient algorithm for multicore partitioned systems. It can be applied to real-time systems with different

criticality levels or those in which only one criticality level exists. This algorithm consists of an allocation technique based on well-known bin packing algorithms that take into account the different frequencies at which a core can operate.

- In addition to the previous algorithm, different strategies for MCS are combined: dropping partitions that are not mandatory in extreme low power situations and reducing the bandwidth of mandatory LO partitions. As a consequence of previous techniques, different energetic plans are obtained, each of them with an energy consumption and performance loss characterization, so at run time the system has to decide to switch to a more energy conserving profile depending on the power sensor values.

- Development of a simulation tool that includes partition to cores allocation as well as the implementation of previous techniques. Thanks to this tool, it is observed that, in the general method (non MCS), an energy saving up to a 5% is achieved. In the extension to MCS, a 35% saving is achieved at the expense of losing performance of LO partitions.

- Implementation on energy-aware issues to a commercial analytic tool, *Xoncrete*. This tool helps the system integrator to build and analyse the schedulability of a partitioned system and, then, to generate a static cyclic plan for the Xtratum hypervisor. All novelties related to energy, as frequency of cores, computation times of tasks depending on frequencies, criticality of tasks, etc. have been implemented to the tool. As output of the tool, a temporal plan with different profiles is obtained.

After this results achieved, it is considered that a theoretical basis is essential for the validation of previous heuristic. Then, as regards the latter, the contributions related to theoretical aspects are:

- Study and analysis of the relation between temporal cost and frequency. Almost all works assume that the relation between time and the inverse of frequency is linear. After measurements of the execution of different functions at different frequencies on a real platform, a model that best relates frequency and time is proposed.

- Definition of a mathematical model of energy consumption, frequency and computation time (utilization). The goal is to find the points (frequency, utilization) that minimizes energy consumption. This is very useful for system designers in order to select between different frequencies that provide the same energy consumption.

- To provide optimal results of energy consumption, i.e., maximum and minimum. It is mathematically demonstrated that when the system runs at its maximum frequency, it consumes the maximum energy and when the system runs at its critical frequency, it consumes the minimum energy.

These later contributions were developed in the Real-Time Systems group at University of Kaiserslautern, Germany, during a predoctoral research stay.

5.2 Future Work

Even though the main objectives have been achieved along this thesis, there are still many possible enhancements and extensions. In this section, future work lines in order to expand the results of this thesis are commented.

Regarding to the generation of offline scheduling plans, two schedulable functions are obtained, $msbf_\tau(t)$ and $gsbf_\tau(t)$. All the calculus is made before the execution, i.e., focused on offline scheduling, due to Xtratum needs. Instead, further work will focus on providing online algorithms to calculate these functions in a specific window. This could be especially useful in flexible environments, where even if the scheduling algorithm is known a priori, jitter or latencies make final slots execution difficult to predict.

Moreover, improvements in the $msbf_\tau(t)$ and $gsbf_\tau(t)$ calculation are also foreseen. The proposed algorithm needs all the hyperperiod space to be exact. This value could be a large number so an upper bound for $msbf_\tau(t)$ and $gsbf_\tau(t)$ might be obtained to avoid studying the complete hyperperiod.

Finally, as all this work considers dynamic priorities in the local level, it may be of interest to develop this study to consider fixed priorities in the local level.

With regard to energy-aware considerations, certain simplifications have also been made. Future work will treat aspects as core overheads due to changes in the system frequency, shared resources and communication channels between partitions, etc.

Moreover, techniques to find optimal solutions of energy consumptions are also desired. In spite that this work proposes CP techniques to solve the problem, it has been observed that the resulting solutions do not always satisfy the temporal requirements. In this sense, future work will be focused on mixed integer linear programming (MILP) models to provide the optimal scenario for energy-aware purposes.

To conclude, we are also considering to extend the study to address the scheduling problem of the system under thermal-aware design. As technology advances, techniques to manage heat dissipation in microprocessors are drawing attention. Thanks to the schedulable supply functions defined in Section 3, the minimum time that the system needs to complete its execution is obtained. Then, the usage of idle time of the processor in order to keep the system temperature between specified

parameters could be a interesting consideration.

These research lines would complement the results and contributions of the present work and help enabling the development of new scheduling techniques for partitioned and hierarchical systems.

5.3 Publications and Projects

In this section, publications and projects in which this thesis has been framed are presented. Figure 5.2 relates the published articles with the contribution chapters of this document.

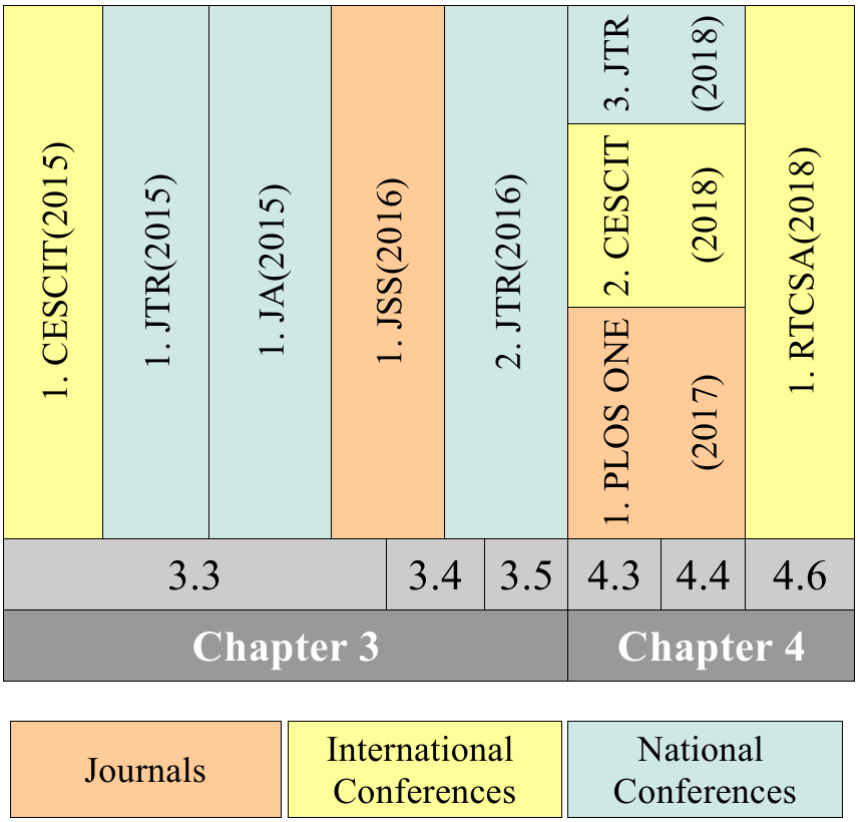


Figure 5.2: Publications related with this thesis

5.3.1 Journals

1. **Title**: Real-time hierarchical systems with arbitrary scheduling at global level

   Authors: Ana Guasque, Patricia Balbastre, Alfons Crespo

   Journal: Journal of Systems and Software

   Editorial: Elsevier (ISSN 0164-1212)

Rank: Q1

Impact Factor: 2,278

Issue: 119 **Pages**: 70-86

Date Reception: February 2016 **Date Publication**: May 2016

2. **Title**: Energy efficient partition allocation in mixed-criticality systems

   Authors: Ana Guasque, Patricia Balbastre, Alfons Crespo

   Journal: PLOS

   Editorial: PLOS ONE

   Rank: Q1

   Impact Factor: 2,766

   Issue: 119 **Pages**: 70-86

   Date Reception: July 2017 **Date Publication**: April 2019

5.3.2 International Conferences

1. **Title**: Schedulability Analysis of Hierarchical Systems with Arbitrary Scheduling in the Global Level

   Authors: Ana Guasque, Patricia Balbastre, Vicent Brocal, Alfons Crespo

   Conference: 2nd IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control (CESCIT 2015)

   Editorial: IFAC. Papers Online 48-10

   Place: Maribor, Slovenia **Date**: 24/06/2015

2. **Title**: Energy efficient partition allocation in partitioned systems

   Authors: Ana Guasque, Patricia Balbastre, Alfons Crespo, Javier Coronel

   Conference: 3rd IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control (CESCIT 2018)

   Editorial: IFAC. Papers Online

   Place: Faro, Portugal **Date**: 08/06/2018

3. **Title**: Energy characterization of real-time partitioned systems.

   Authors: Ana Guasque, Patricia Balbastre, Alfons Crespo, Gerhard Fohler

   Conference: 24th IEEE International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA 2018)

   Editorial: IEEE Xplore

   Place: Hakodate, Japan **Date**: 31/08/2018

4. **Title**: Design of Criticality-Aware Scheduling for Advanced Driver Assistance Systems

   Authors: Savithry J, Ana Guasque, Anju Pillai, Patricia Balbastre, Alfons Crespo

   Conference: 24th International Conference on emerging technologies and factory automation (ETFA 2019)

   Editorial: IEEE Xplore

   Place: Zaragoza, Spain **Date**: 12/09/2019

5.3.3 National Conferences

1. **Title**: Schedulability Analysis of Hierarchical Systems with Arbitrary Scheduling in the Global Level

   Authors: Ana Guasque, Patricia Balbastre, Vicent Brocal, Alfons Crespo

   Conference: XVIII Jornadas de Tiempo Real (JTR 2015)

   Place: Murcia, Spain **Date**: 29/01/2015

2. **Title**: Análisis de planificabilidad de sistemas jerárquicos con planificación arbitraria en el nivel global

   Authors: Ana Guasque, Patricia Balbastre, Vicent Brocal, Alfons Crespo

   Conference: XXXVI Jornadas de Automática (JA 2015)

   Editorial: Comité Español de Automática de la IFAC (CEA-IFAC)

   Place: Bilbao, Spain **Date**: 02/09/2015

3. **Title**: Definición de funciones de asignación de CPU planificables para sistemas jerárquicos con planificación arbitraria en el nivel global.

   Authors: Ana Guasque, Patricia Balbastre, Alfons Crespo

   Conference: XIX Jornadas de Tiempo Real (JTR 2016)

   Place: Málaga, Spain **Date**: 04/02/2016

4. **Title**: Algoritmo de asignación de particiones para la eficiencia energética de sistemas particionados

   Authors: Ana Guasque, Patricia Balbastre, Alfons Crespo, javier Coronel

   Conference: XX Jornadas de Tiempo Real (JTR 2018)

   Place: Bilbao, Spain **Date**: 24/01/2018

5.3.4 Projects, Scholarships and Research Stay

Fig. 5.3 relates same publications with the research projects, scholarship and research stays, which have been framed by the development of this thesis.

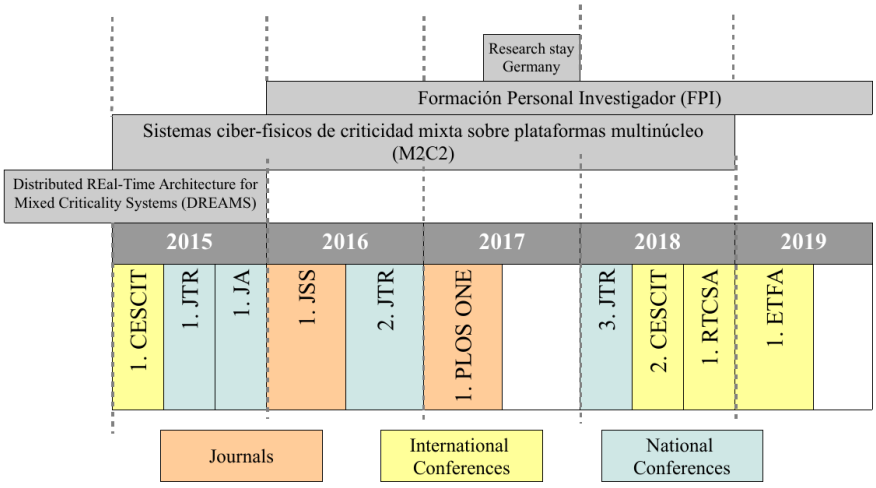


Figure 5.3: Publications over time and related projects

1. **Project**: Open and cost-effective virtualization techniques and supporting separation kernel for the embedded systems industry (VOS4ES)

Reference: 286706

Duration: 2011-2013

2. **Project**: Distributed REal-Time Architecture for Mixed Criticality Systems (DREAMS)

   Reference: 610640

   Duration: 2013-2015

3. **Project**: Sistemas ciber-fisicos de criticidad mixta sobre plataformas multi-núcleo (M2C2)

   Reference: TIN2014-56158-C4-1-P

   Duration: 2015-2018

4. **Schoolarship**: Formacion de Personal Investigador (FPI)

   Center: Universitat Politècnica de València

   Reference: BES-2015-072273

   Duration: 2016-2019

5. **Research Stay**: Real-Time Systems Group

   Center: Technische Universität Kaiserslautern, Germany.

   Supervisor: Prof. Gerhard Fohler

   Duration: 09/2017-12/2017

Appendices

APPENDIX

A

Supplementary calculation

A.1 Interpolating experimental values through a rational function

In mathematics, Thiele's interpolation formula is the way to define a rational function $f(x)$ from a finite set of inputs x_i and their function values $f(x_i)$. It is expressed as a continued fraction, where ρ represents the reciprocal difference:

$$f(x) = f(x_1) + \frac{x - x_1}{\rho(x_1, x_2) + \frac{x-x_2}{\rho(x_1,x_2,x_3)-f(x_1)+\frac{x-x_3}{\rho(x_1,x_2,x_3,x_4)-\dots}}} \tag{A.1}$$

In order to calculate the rational function through Thiele's method, we have to use the Equation A.1 and experimental values in Table 4.37. Firstly, we have to define $\varphi[x_0, x_1]$ as the reverse difference obtained from known points.

$$\varphi[x_0] = f(x_0) = y_0 = 0.4$$
$$\varphi[x_0, x_1] = \frac{x_0 - x_1}{y_0 - y_1} = -0.15$$
$$\varphi[x_0, x_2] = \frac{x_0 - x_2}{y_0 - y_2} = -0.333$$
$$\varphi[x_0, x_1, x_2] = \frac{x_1 - x_2}{\varphi[x_0, x_1] - \varphi[x_0, x_2]} = -0.382$$

$$\varphi\left[x_0, x_3\right] = \frac{x_0 - x_3}{y_0 - y_3} = -0.8$$
$$\varphi\left[x_0, x_1, x_3\right] = \frac{x_1 - x_3}{\varphi\left[x_0, x_1\right] - \varphi\left[x_0, x_3\right]} = -0.3841$$
$$\varphi\left[x_0, x_4\right] = \frac{x_0 - x_4}{y_0 - y_4} = -1.0277$$
$$\varphi\left[x_0, x_1, x_4\right] = \frac{x_1 - x_4}{\varphi\left[x_0, x_1\right] - \varphi\left[x_0, x_4\right]} = -0.387$$

Secondly, we calculate the table of reverse differences (see Table A.1).

Table A.1: Reverse differences table

x_k	y_k	$\varphi[x_0, x_k]$	$\varphi[x_0, x_1, x_k]$
0.07	0.4		
0.1	0.2	-0.15	
0.17	0.1	-0.333	-0.382
0.35	0.05	-0.8	-0.3841
0.44	0.04	-1.0277	-0.387

As $\varphi[x_0, x_1, x_k]$ converts in -0.38, the calculus stops and this value is used to calculate the desired function (for all points in Table 4.37):

$$\begin{aligned} f(x) &= 0.4 - 0.385\frac{x - 0.07}{x} = \\ &= \frac{0.4x - 0.385x + 0.02688}{x} = \\ &= \frac{0.016x + 0.02688}{x} \end{aligned}$$

A.2 Evaluation of the character of a relative extrema

To know if the relative extremum (U_0, f_0) is a maximum or a minimum, we will apply the second partial derivate test. The second partial derivative test tells us how to verify whether this stable point is a local maximum, local minimum, or a saddle point[1]. Then, we can build a symmetric square matrix called Hessian matrix of $\mathcal{L}$, composed of second-order partial derivatives of this function.

[1]By definition, saddle points are stable points where the function has a local maximum in one direction, but a local minimum in another direction.

$$H = \begin{bmatrix} \frac{\partial^2 \mathcal{L}}{\partial U^2} & \frac{\partial^2 \mathcal{L}}{\partial U \partial f} \\ \frac{\partial^2 \mathcal{L}}{\partial U \partial f} & \frac{\partial^2 \mathcal{L}}{\partial f^2} \end{bmatrix} \tag{A.2}$$

Then:

- If $H(U_0, f_0)$ is a positive-definite matrix, then $E(U, f)$ will have a conditioned local minimum in (U_0, f_0).
- If $H(U_0, f_0)$ is a negative-definite matrix, then $E(U, f)$ will have a conditioned local maximum in (U_0, f_0).
- If $H(U_0, f_0)$ is neither positive nor negative, we will need another method to characterize the conditioned point.

Consequently, the Hessian matrix of $\mathcal{L}$ is:

$$H(U, f) = \begin{bmatrix} \frac{2\lambda k}{U^3} & MAF \cdot \beta \cdot \alpha \cdot (\alpha - 1) \cdot U \cdot f^{\alpha - 2} \\ MAF \cdot \beta \cdot \alpha \cdot (\alpha - 1) \cdot U \cdot f^{\alpha - 2} & MAF \cdot \beta \cdot \alpha \cdot f^{\alpha - 1} \end{bmatrix} \tag{A.3}$$

Applying this general definition to the critical point (U_0, f_0) and with the previously defined values α, β, P_s and *MAF*, the Hessian matrix of $\mathcal{L}$ is:

$$H(U_0, f_0) = \begin{bmatrix} \frac{3k}{250 \cdot 8^3 \sqrt[3]{20}} & \frac{3}{2} \sqrt[3]{\frac{2}{250}} \\ \frac{3}{2} \sqrt[3]{\frac{2}{250}} & 6 \sqrt[3]{\frac{4}{25}} \end{bmatrix} \tag{A.4}$$

So then, as the matrix H is positive definite, the critical point $H(U_0, f_0)$ is a minimum of E.

List of Figures

List of Tables

Statement of Originality

I, Ana Guasque Ortega, do herewith declare that the material contained in my thesis entitled "Study, analysis and new scheduling proposals in partitioned real-time systems" is original work performed by me under the guidance and advice of my faculty advisors Patricia Balbastre Betoret and Alfons Crespo Lorente. I have read and do understand the "Documento de compromiso de elaboración de tesis doctoral en la Universitat Politècnica de València". By signing this statement I unequivocally assert that this thesis conforms the policies.

This thesis was presented in Valencia at December 2019

White page